Exploring Dance Expression Through Self-Supervised Transformer-Based Contrastive Representation Learning

Samuel Tong
Stanford University
samtong@stanford.edu

Abstract

In this project, I explore modeling dance expression in video through self-supervised contrastive representation learning. I specifically focus on learning a model that projects K-Pop (Korean Pop) dance group videos into a high-dimension embedding space to encode group dance expression, as well as choreography-specific (video-specific) dance expression. In this report, I detail the end-to-end process of dataset preprocessing and collection, feature extraction, and model architecture. I end with an analysis, focusing in part on the implications and limitations of scale within this project as well as possible future work.

1. Introduction

The intersection of dance and computer vision heavily emphasizes the objective of motion recognition and analysis. But from an artistic lens, dance is much more than pure motion: it is a human-centric form of expression, style, and creativity. In some sense, movement classification reduces dance down to a detached, highly logical objective, perhaps ignoring some of the inherent emotion and feeling that is central to dance as an art.

In this project, I propose shifting away from classification of dance *movement*, and instead a turning towards an objective of comparing relative dance *expression*. While a somewhat broad term, I refer to expression as the abstract, human-centered tendencies within a dance performance that drives our conceptualized identity of a performance or performer. Expression, in that sense, differs from pure "movement" in that it is not a specific sequence of motions; it is how those motions are embodied and expressed and brought to life.

In this project, I utilize contrastive representation learning to create a model that learns embedding vector of a dance's movement expressions with respect to a dancer or group of dancers (analogous to learned token embeddings for large language models). I focus specifically on learned representations within K-Pop (Korean Pop) dance. In addition to growing as a popular global phenomenon in recent years, K-Pop dance is a fusion of various dance styles, borrowing from genres including hip hop, contemporary, jazz, etc. To that end, K-Pop dance is not discretely classifiable into a single existing genre of dance movement, thus providing nuance and variance in its style. Further, each K-Pop group (generally comprised of multiple members) carries their own unique, differentiating dance style and expression, creating a grounded "identity" of the group and its members with respect to their dance performance.

As a concrete objective, the model should be able to take a new, unseen video example and output an embedding representation that can be compared in similarity to embedded dataset examples, both at a video level (e.g. "this person dances in the style of [dance video] by [K-Pop group]"), as well as at a group level (e.g. "this person generally dances in the style of [K-Pop group]"). Note that the phrase "in the style of" is inherently abstract and somewhat ambiguous; indeed, it could encompass a range of factors including the stylistic tendencies of the dancers (both K-Pop group members, as well as any backup dancers that appear in the video), the choreographer, the genre of the dance choreography, etc. Part of the curious nature of this project is seeing what aspects of the video examples the model will learn to differentiate, embed on, and prioritize.

2. Related Work

Dance movement has been a widely studied domain in context of deep learning and computer vision, with objective tasks ranging from choreography creation to dance performance analysis [2]. One prominent objective is movement classification, in which models are trained to classify dance movement given a video example. Previous works

in this domain explore various architectural implementations: for example, [6] classifies movement by extracting features with a multi-layer convolution, then passing features through an LSTM architecture. [3] takes a different approach, using a grayscale and background subtraction pipeline to generate dancer silhouettes, which are passed through an unsupervised self-organizing map neural network. Further, [7] and [8] both attempt to predict dancers' Laban movements (a general notation for describing human movement [11], analogous to sheet music notation for musical performance), but their architectural implementations differ. Namely, [7] uses pose estimation as features then passes through an attention and CNN layer, while [8] combines human body fitting (via pose estimation) and point cloud floor estimation to predict Laban movements.

3. Dataset and features

To my knowledge, there are no K-Pop specific video datasets to meet the needs of this project. Additionally, existing dance datasets (such as AIST++), while extensive and large, do not meet the desired requirements of a collective group "identity" expression throughout video examples. To this end, I collected my own dataset by scraping K-Pop dance practice videos from YouTube.

My dataset contains video examples from YouTube from the following K-Pop groups: BTS, ATEEZ, Stray Kids, NCT 127, ENHYPEN, and TOMORROW X TOGETHER. Conveniently, these groups provide official curated YouTube playlists of all prior dance practices, which I use to determine which videos to scrape. Since some videos within playlists are formatted as YouTube Shorts (which generally showcase small snippets of a full dance practice video for promotional purposes), I filter videos from the playlists that do not meet a minimum time length of at least 1 minute and 15 seconds (since dance practice videos match the length of their corresponding song, full videos are generally least 2 minutes in duration).

I scraped dataset videos using the Python library `yt-dlp` and stored videos as mp4 files in a private AWS S3 bucket. The dataset contains approximately 300 videos for a total of 19 hours of video. Each video example is stored in full as an mp4 file, and has a corresponding JSON metadata file listing the video title, author, length, and playlist from which it was sourced.

4. Methods

For this model, I use keypoints extracted via pose estimation as feature extraction. The primary motivation for this approach is to encourage the model to focus primarily on the dance movement, rather than extraneous information. In particular, utilizing pose estimation instead of RGB pixel values obfuscates the dancer's identity, forcing the model to learn based off of dancer movement rather than memorizing physical appearances.

4.1. Sampling

For a given video, I uniformly sample examples (that is, extracted keypoints for a sequence of frames) as follows. Let $t = 15$ denote the total number of desired sampled seconds per video, and let $l = 2$ denote the (upper bounded) desired seconds per example. I uniformly sample from $\lfloor t/l \rfloor = 7$ regions per video (again, each of length l). I additionally add a constant padding of $p = 3$ seconds both at the beginning and end of the video to offset uniform sampling, such that the sampling avoids potentially empty space with no dance movement at the beginning and end.

Most videos in the dataset include multiple dancers (note that the K-Pop groups used in dataset collection range from five to ten total members). For a given region (i.e. sequence of frames) we extract multiple examples, one per visible dancer. Examples are expected to be contiguous (i.e. non-disjoint), starting from the first frame of the sampled region – that is, if a given dancer cannot be tracked temporally within the example for a frame (e.g. they are blocked by another dancer), we immediately stop further sampling, and no future entries are appended to the sequence (even if the dancer is visible again).

Let f denote a given video's frame rate (that is, frames per second). Because frame rates match the original YouTube video frame rate and thus vary by video, we sample every s frames, where $s = \max\{1, \lfloor f/30 \rfloor\}$. In particular, we see that for $f < 60$, $s = 1$ (i.e. we sample every frame), but for $60 \leq f < 120$, $s = 2$ (i.e. we sample every other frame). This ensures consistency such that the model maintains rough temporal consistency within any given sequence of frames [1]. In total, each example has up to (assuming the example is fully contiguous) $(f \cdot l/s)$ frames; given that $l = 2$, a given example has up to 60 frames.

Finally, examples are labeled by both group ID (an alias of the YouTube playlist from which the example's video was derived), and a video ID (the YouTube video ID from which the example was taken).

[1] Here, we operate the assumption that the temporal differences between, say, $f = 24$ and $f = 30$ can be trivially learned by the model. Also note that the speed with which a given dance is executed is dependent on the tempo of the music, and we assume that the difference between frames should be trivial in context of this additional variance in speed.

4.2. Keypoint Extraction

For keypoint extraction, I use ViTPose[13], a transformer-based pose estimation model. Because ViTPose expects the person to be fully in frame at inference time, I first pass a given frame through DETR[1], a transformer-based object detection model, to obtain bounding boxes for each dancer in frame.[2] DETR bounding boxes were retained given a minimum score threshold of 0.75, and tracked per dancer temporally using ByteTrack[14]. Each example is outputted as a tensor of shape `(num_frames, num_keypoints, 2)`, where num_keypoints = 17 (derived from the output of ViTPose), and `(:, :, 0)` and `(:, :, 1)` denote the x and y pixel coordinates, respectively.

This process yielded a total of 12,925 examples. In order to minimize noise, I filtered examples that did not meet the following requirements: 1) all keypoints across frames for a given example must have a confidence score (extracted from ViTPose) of at least 0.15; 2) the mean confidence score across all keypoints in an example must be at least 0.5; and 3) the total number of frames for an example must be at least 18 (approximately 0.75 seconds at 24 FPS). Among the 12,925 examples, the filtering process yielded a total of 5680 valid examples, for a retention rate of 0.439. Of the filtered examples, 43.8% were filtered for their lowest keypoint score (1), 0.4% for their mean confidence score (2), and 43.6% for the total number of frames (3). Notably, 12.5% were additionally filtered for exceeding 60 total frames; I hypothesize that this is because the average video FPS (computed using the Python library OpenCV) was computed to be slightly under 60 FPS (e.g. 59 frames per second).

Finally, for a given example, I normalized each keypoint value (i.e. the raw x/y pixel locations) relative to the value of keypoint 16 (the last keypoint, corresponding to the right foot) at the first frame, such that the tensor value at `(0, 16, :)` is `(0, 0)`. This obfuscates the location of the dancer within the entire frame, encouraging the model to disregard this information as it learns.

4.3. Model

The model is given examples as tensors of size $(\text{num\_frames}, \text{num\_keypoints}, 2)$. Again note that an example can have up to 60 frames; to account for this variation, the remaining entries between num_frames and the maximum 60 frames are masked with zeros.

[2] I use the HuggingFace Transformers[12] models `usyd-community/vitpose-base-simple` and `PekingU/rtdetr_r50vd_coco_o365` for DETR and ViTPose, respectively.

The model first flattens examples into size $(\text{num\_frames}, \text{num\_keypoints} \cdot 2)$, then passes the frames through a linear transformation into the transformer embedding space. From there, frames are attenuated via a transformer block. The transformer output is finally average pooled across frames, then passed through a final fully connected layer to the embedding space.

Let x_i denote the model output, and y_{g_i}, y_{v_i} denote the group label and video id label, respectively. Then, for a single batch $\mathcal{B}$ of size N, I utilize the following loss function (adopted from [10]):

$$\mathcal{L}_{\mathcal{B}} = \sum_{i=1}^{N-1} \sum_{j=i+1}^{N} w \cdot L(x_i, x_j, y_{g_i}, y_{g_j}) + (1-w) \cdot L(x_i, x_j, y_{v_i}, y_{v_j})$$

where $w \in [0, 1]$, and L is defined as follows:

$$L(x_i, x_j, y_i, y_j) = \mathbf{1}[y_i = y_j] \|x_i - x_j\|_2^2 + \mathbf{1}[y_i \neq y_j] \max(0, \epsilon - \|x_i - x_j\|_2)^2$$

In other words, $\mathcal{L}_{\mathcal{B}}$ computes the loss between every combination of pairs of outputs in the batch. In particular, note that we weight the loss of group labels and video id labels by w and $(1-w)$, respectively, where w is a tunable hyperparameter. Note that while the two objectives (optimizing for group label, and optimizing for video label) are not two completely distinct tasks (since a correct video label implies a correct group label), intuitively this weighting allows us to specify which aspect of embedding representation learning the model should focus on.

While positive-pair losses (e.g. CLIP loss) are widely used, I chose to utilize this contrastive loss function as the dataset does not immediately lend itself to positive and negative pairs. In particular, because the model does not perform augmentations on examples, examples are not grouped into pairs. While directly creating positive and negative pairs and passing each into a batch is theoretically possible, it is cumbersome with this architecture given that it would be necessary to link pairs of examples per video (which further implies that we will likely not see all the data in one epoch if some video contain more corresponding examples than others).

5. Experiments, Results, and Discussion

5.1. Metrics

In addition to the contrastive loss function defined above, I measure the top-3 accuracy of the embedding outputs

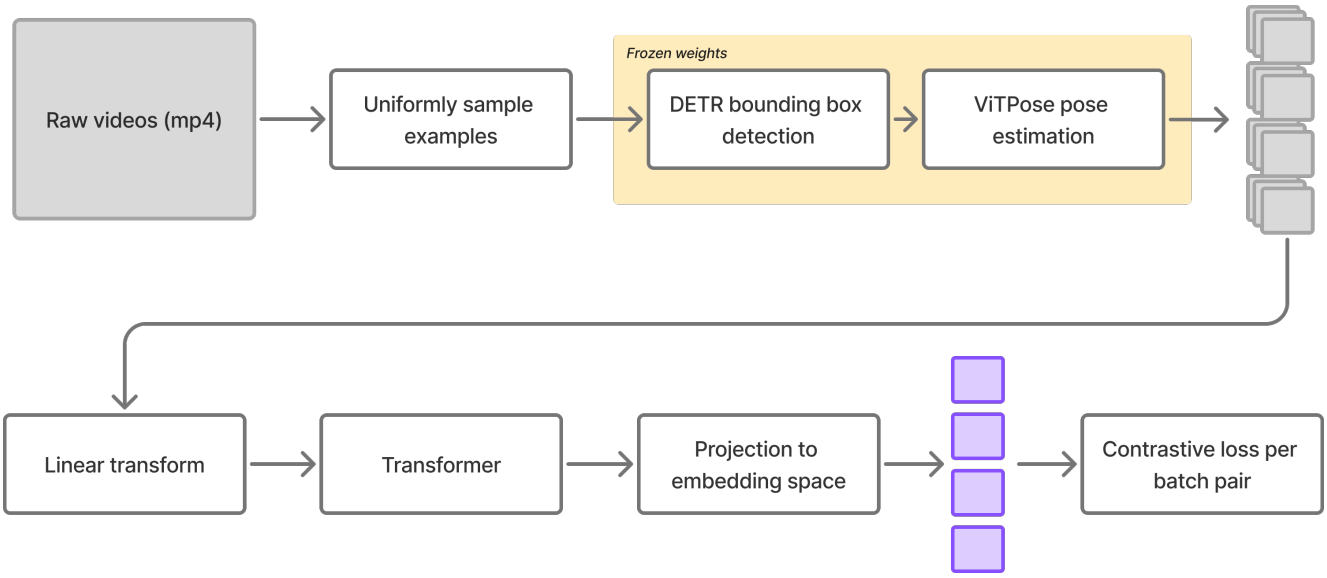

Figure 1. Model architecture. Raw videos are uniformly sampled, then passed through the DETR and ViTPose models to extract bounding boxes and pose estimation, respectively. This preprocessing pipeline yields a batch of sequences of frames, where each sequence entry contains the extracted x/y keypoint coordinates. These frames are passed through a linear transform, a transformer architecture, and finally outputted to the final embedding space.

with respect to both group and video id labels. Formally, let y_{g_i}, y_{v_i} denote the ground truth group and video id labels for example x_i. Then, let $Y_{g_i} = \{y_{g_a}, y_{g_b}, y_{g_c}\}$ and $Y_{v_i} = \{y_{v_a}, y_{v_b}, y_{v_c}\}$ denote the group and video id labels (respectively) of the 3 closest embedding vectors to x_i, determined via cosine similarity. Then, we define the top-3 accuracy as follows:

$$\text{Acc}_g = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[y_{g_i} \in Y_{g_i}]$$

$$\text{Acc}_v = \frac{1}{N}\sum_{i=1}^{N} \mathbf{1}[y_{v_i} \in Y_{v_i}]$$

Hyperparameter	Value	Description
init_lr	1e-4	
eps	20	ϵ in loss eq.
loss_weighting	0.33	w in loss eq.
num_epochs	30	
batch_size	16	
transformer_dim	1024	
transformer_nheads	8	
transformer_ff_dim	4096	
transformer_nblocks	8	
transformer_dropout	0.1	
out_embed_dim	1024	

Table 1. Baseline hyperparameters

5.2. Experiments and Results

I first conducted an informal, preliminary hyperparameter search using Weights and Biases sweeps feature to determine the approximate range of appropriate hyperparameters. Using this information, I then trained a baseline model (see Table 1), using k-fold cross-validation ($k = 5$). Note that each fold was shuffled and partitioned along video ID, such that all examples corresponding to a single video ID were placed in the same batch. This was done to avoid leaking data from one batch to another, since examples sampled from the same video region could have multiple dancers performing the same move. While this implies that each batch may be of a different size (given that there are a varying number of examples corresponding to a single video), in practice the batch sizes were relatively consistent[3]

Results of the baseline experiment are shown in Table 2, and Figures 2, 3, and 4. Note in particular that while the training loss decreased, validation loss converged to an order of magnitude higher than the training loss. In addition, accuracies were predominantly constant and did not improve over time.

As a qualitative metric, I ran PCA with $n = 2$ dimensions to see if any visual patterns were emergent in the final iteration. Specifically, I passed all examples (from both the

[3]In particular, batches were of sizes 1150, 1114, 1107, 1169, and 1140.

Metric	Train	Val
Loss	3.19e+3	4.40e+4
Top-3 Acc (group)	43.3%	44.0%
Top-3 Acc (video)	1.86%	6.97%

Table 2. Baseline results, averaged across folds

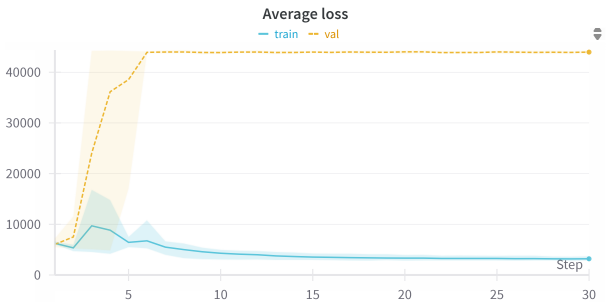

Figure 2. Baseline average loss

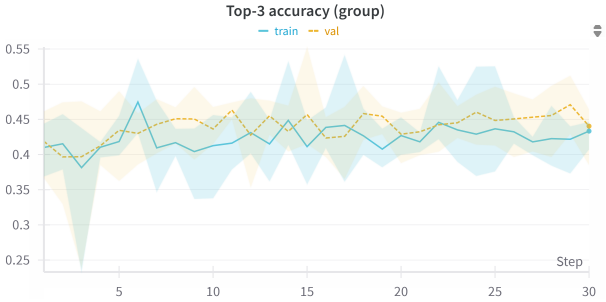

Figure 3. Baseline top-3 accuracy (group)

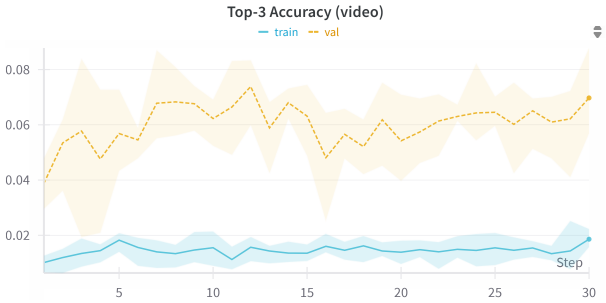

Figure 4. Baseline top-3 accuracy (video)

train and validation dataset) through the trained model, then fit the resulting vectors via PCA, labeled by group. The results are shown in 5; while there appear to be two trend directions (one moving horizontal, and the other moving up and to the right), there is no clearly distinctive decision boundary between groups as we would have hoped. There are, however, two relevant observations. Firstly, (and perhaps most immediately apparent), there is one large outlier

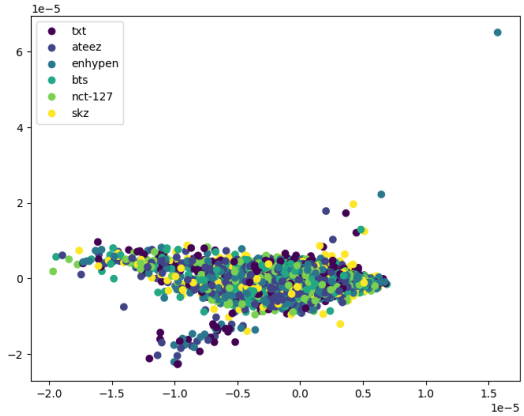

Figure 5. PCA analysis ($n = 2$), colored by group. For brevity, in the legend I abbreviate group "TOMORROW X TOGETHER" as "txt," and "Stray Kids" as "skz."

belonging to the group "ENHYPEN." While it is not particularly apparent why it is so far removed, it follows the trend direction of points moving up and to the right. Second, the bottom-middle range of the scatterplot (approximately $x = -0.75, y = -1.5$) seems to be a cluster that only includes three of the six groups (note that BTS, NCT 127, and Stray Kids are not present), indicating that these examples from the present groups are particularly similar. Perhaps these examples contained similar dance movements; or, examples in this cluster belonging to the same group potentially belonged to the same region of the same sampled video.

5.3. Hyperparameter Tuning

Evidently, the baseline model in its current state was not performant. To this end, I conducted subsequent hyperparameter tuning experiments; in particular, I was interested in the loss weighting hyperparameter w, and the relative importance of prioritizing group loss ($w \to 1.0$) versus video ID loss ($w \to 0.0$). In the baseline, I set $w = 0.33$, hypothesizing that slightly weighting video ID loss (a more difficult objective) over group loss would be advantageous. In Table 3, I describe subsequent experiments with varying values of w, with experiment trained for 15 epochs and without k-fold cross-validation for efficiency.

From the validation set results, it appears that as loss weighting increases, the model generally performs marginally better. However, given that the validation dataset is relatively small (1000 examples), these metrics are not robustly sound; to confidently verify, we would need more data. It is also important to note that these improvements are, again, marginal.

loss_weighting	Loss	Group Acc*	Video Acc*
0.0	4.6e+4	43.6%	4.9%
0.33 (baseline)	4.2e+4	43.3%	5.3%
0.66	4.1e+4	43.3%	5.3%
1.0	3.9e+4	45.3%	6.6%

Table 3. Hyperparameter tuning results. All metrics are in terms of validation data.
*I calculate this with an exponential moving average (with a coefficient of 0.5) as these values have especially high variance.

It is worth noting that I conducted additional informal experiments on the side, including increasing transformer dimensions, increasing the loss epsilon hyperparameter, and doubling the batch size. However, none of these experiments resulted in additional improvements that would merit any further discourse.

5.4. Discussion

It appears that, while the model learned to reduce training loss over time, all other metrics stayed approximately constant and did not improve. While this is somewhat disappointing, there are many things we can still observe. As mentioned previously, adjusting the loss weighting w towards 1.0 resulted in some marginal improvement. It is interesting that emphasizing group accuracy (a higher w) also improved the model's video accuracy; intuitively, perhaps the group accuracy is a somewhat viable proxy for improving video accuracy (a relatively harder task to optimize, especially given the small dataset size).

Interestingly, the Top-3 video accuracy scored higher on the validation than training dataset. This is likely because examples were partitioned in folds by videos; thus, a validation batch would be more likely to contain examples with the same video id than a training batch of the same size. Unfortunately, this is a suboptimal result of the contrastive loss function design choice, as we do not inherently have positive pairs and need to source them from somewhere. We also note that the alternative (allowing examples from the same video to be spread across batches) could result in data leakage (again, since examples sampled from the same region may have dancers performing the same dance), which is not any more desirable.

While the results of the project did not show convergence to a particularly insightful outcome, it is of interest to discuss why this occurred. I hypothesize that the quantity of data is likely the main bottleneck in model performance (roughly, 5000 examples at two seconds each is approximately 2 hours and 45 minutes of examples – a relatively small dataset). Given more time and resources, I would have hoped to collect more data (either at the video scraping stage, or during example sampling). A large inhibitor to scaling up data collection was DETR and ViTPose inference: each frame took on the order of 0.1s to process through both models. While this would be reasonably efficient at a small scale, this would have been much more intensive to undertake with respect to time and compute at a large scale (e.g. by this metric alone, collecting one million examples would take at least 28.8 total hours – and this is before filtering). Indeed, it took around 7 hours in total to collect the initial 12,000 examples. The quantity of initial videos scraped from YouTube, too, were also limited, as YouTube's bot detection algorithm blocked video downloads on EC2 servers; my workaround was to download videos locally on my personal computer, then upload back to a cloud server. But this too was slow and expensive. Perhaps these challenges are excellent learning opportunities to help understand the intricacies and complexities of effective data collection at scale.

Some other limitations to note: because I defined clip sampling regions by seconds (i.e. a desired 2 seconds per clip) rather than by number of frames, the model might have internalized some examples to be temporally faster or slower than they actually were. As mentioned previously, I hypothesized that 1) this would not make a marginal difference as there is inherently variation in song tempo (and subsequently, dance speed), and 2) even if it did make a marginal difference, the transformer architecture would be able to learn to ignore this. In hindsight however, the latter hypothesis would be contingent on a sufficient amount of data, which was not adequately presented to the model.

More broadly beyond the pure outcomes of the experiments, the process of creating an end-to-end pipeline from data collection to fine-tuning was in itself an incredibly insightful learning experience. While the results may not have been what I had hoped, undertaking the (somewhat laborious) process of scraping and curating data, determining model architecture, and attempting to finetune hyperparameters pushed me to learn a tremendous amount.

6. Conclusion and Future Work

In this report, I outline my process of an end-to-end pipeline to create a transformer-based contrastive representation learning model, explaining my process and reflections from data collection, feature extraction, and hyperparameter tuning. While the model did not show particularly satisfying results (I hypothesize in large part due to the scale of the dataset), the project provides a baseline framework that can be used and scaled by future research (additionally, time permitting, I also hope to conduct additional experiments before the poster session to achieve better perfor-

mance). As a whole, the intersection of dance expression and deep learning is a unique application; within this space, there is still much more to be learned, and much more potential to be gleaned.

7. Contributions

All work was conducted by myself. A special thanks to the TA's for their invaluable guidance.

References

[1] N. Carion, F. Massa, G. Synnaeve, N. Usunier, A. Kirillov, and S. Zagoruyko. End-to-end object detection with transformers, 2020. 3

[2] M. R. Nogueira, P. Menezes, and J. M. de Carvalho and. Exploring the impact of machine learning on dance performance: a systematic review. *International Journal of Performance Arts and Digital Media*, 20(1):60–109, 2024. 1

[3] Y. Pang and Y. N. and. Dance video motion recognition based on computer vision and image processing. *Applied Artificial Intelligence*, 37(1):2226962, 2023. 2

[4] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Köpf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai, and S. Chintala. Pytorch: An imperative style, high-performance deep learning library, 2019.

[5] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, A. Müller, J. Nothman, G. Louppe, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and Édouard Duchesnay. Scikit-learn: Machine learning in python, 2018.

[6] W. Qin and J. Meng. The research on dance motion quality evaluation based on spatiotemporal convolutional neural networks. *Alexandria Engineering Journal*, 114:46–54, 2025. 2

[7] D. Sun and G. Wang. Deep learning driven multi-scale spatiotemporal fusion dance spectrum generation network: A method based on human pose fusion. *Alexandria Engineering Journal*, 107:634–642, 2024. 2

[8] M. Turab, P. Colantoni, D. Muselet, and A. Tremeau. Dance style recognition using laban movement analysis, 2025. 2

[9] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin. Attention is all you need, 2023.

[10] L. Weng. Contrastive representation learning, May 2021. 3

[11] Wikipedia. Labanotation — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Labanotation&oldid=1276233755, 2025. [Online; accessed 16-May-2025]. 2

[12] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush. Huggingface's transformers: State-of-the-art natural language processing, 2020. 3

[13] Y. Xu, J. Zhang, Q. Zhang, and D. Tao. Vitpose: Simple vision transformer baselines for human pose estimation, 2022. 3

[14] Y. Zhang, P. Sun, Y. Jiang, D. Yu, F. Weng, Z. Yuan, P. Luo, W. Liu, and X. Wang. Bytetrack: Multi-object tracking by associating every detection box, 2022. 3