Human Action Recognition

Hesam Mohebi, 7505478, University of Genoa

June 2025

1. Abstract

This report presents a systematic evaluation of computational models for the 60-class action recognition task on the BABEL dataset. [1]. I explore a hierarchy of methods, from classic machine learning models to custom-trained deep sequence models and large-scale pretrained architectures (2s-AGCN[2] , MotionCLIP[3]). My results demonstrate a clear performance progression: the KNN baseline achieved 34.4% accuracy, a custom LSTM improved to 38.0%, and the pretrained MotionCLIP model obtained the highest accuracy of 45.47%. This work concludes that while custom models can capture meaningful dynamics, the complexity and class imbalance of BABEL are best addressed by leveraging the powerful representations of large pretrained models.[1]

2. Introduction

3D skeleton-based Human Action Recognition (HAR) is a crucial computer vision task that offers robustness over traditional video-based methods by focusing purely on body dynamics. A key benchmark in this area is the BABEL dataset, which provides precise 3D motion-capture data with dense, per-frame action labels. Its realistic action categories make it a particularly challenging task for developing generalizable models.

This report presents a systematic evaluation of different models on the BABEL 60-class recognition task. My project provides an exploration across a hierarchy of techniques, beginning with classic machine learning baselines (e.g., KNN) using hand-crafted spatio-temporal features. Then I moved to a custom-trained bi-directional LSTM to learn features. Finally, I assessed the impact of transfer learning by evaluating large-scale pretrained architectures, including the Adaptive Graph Convolutional Network (AGCN) and the MotionCLIP framework. This approach allows for a clear comparison of feature engineering, custom deep learning, and transfer learning paradigms on BABEL.

3. Methodologies

This section introduces the dataset, preprocessing pipeline, and the various models implemented and evaluated for the action recognition task.

3.1 Dataset

This project utilizes BABEL (Bodies, Action, and Behavior with English labels)[1]. For compatibility with action recognition models, the data is converted into NTU RGB+D skeleton format. Each sample contains 5 5-second motion clips recorded at 30 FPS, resulting in sequences of 150 frames[1]. The input data is a (25, 3, 150) sample corresponding to 25 joints, 3 spatial dimensions. (x, y, z), And 150 temporal frames.

[1] Link to project's codes

3.2 Preprocessing

To normalize the data before training, all skeleton sequences undergo a three-step preprocessing pipeline according to the original implementation.[1]:

1. Null Frame Padding: Any missing or null frames within a sequence are filled by repeating the last valid preceding frame.
2. Centering: The skeleton is centered in the coordinate space by subtracting the position of the main body center (joint 1, spine) from all other joints in every frame.
3. Alignment: To create a consistent orientation, the skeleton is aligned to the coordinate axes. The bone connecting the hip (joint 0) and spine (joint 1) is aligned to the Z-axis, and the bone between the left and right shoulders (joints 4 and 8) is aligned to the X-axis.

$$\boldsymbol{X} \in R^{C\times T\times V\times M}, \qquad C = 3(\text{x,y,z}), \qquad T = 150, V = 25, M = 1$$

3.3 Baseline Models and Feature Engineering

To establish a performance baseline, I used classic machine learning models with sets of hand-crafted features extracted from the preprocessed skeleton sequences.

3.3.1 Feature sets:

1. Basic Features: An initial simple representation was created by calculating the mean and standard deviation of each coordinate channel across all joints and frames (Table 1). This resulted in just 6 features, losing nearly all temporal and relational information.
2. Rich Features: To capture dynamics, I calculated motion-based features (Table 1):
   - Velocity: The frame-to-frame displacement of each joint, encoding motion intensity.
   - Acceleration: The change in velocity, capturing the rhythm and sharpness of movements.
   - Bone Vectors: The spatial offset vectors between connected joints (e.g., wrist minus elbow), providing a translation-invariant representation of the body's pose configuration.
3. Extended Features: The "Rich" feature set was augmented with the Euclidean distances between 10-20 selected joint pairs (e.g., wrist-elbow, head-neck) to further detail the relative geometry of the pose (Table 1).

Table 1: Extracted features formula

Block	**Formula**
Basic statistics	$\mu_c = \sum_{t,v} X_{c,t,v}\,, \qquad \sigma_c = \sqrt{\frac{1}{TV}\sum_{t,v}\left(X_{c,t,v} - \mu_c\right)^2}$
Velocity	$\boldsymbol{V_{c,t,v}} = X_{c,t+1,v} - X_{c,t,v}$
Acceleration	$\boldsymbol{A_{c,t,v}} = V_{c,t+1,v} - V_{c,t,v}$
Bone vector	$\boldsymbol{B_{c,t,v}} = X_{c,t,v} - X_{c,t,p(v)}$
Pooling operator	.$\text{Pool}(\boldsymbol{Y}) = [\bar{y}_c,\, s_c]_{c=1}^{3}, \bar{y}_c = \sum_{t,v} Y_{c,t,v}\,, s_c = \sqrt{\frac{1}{TV}\sum_{t,v}\left(Y_{c,t,v} - \bar{y}_c\right)^2}$
Rich feature vector	$f_{rich} = [Pool(X), Pool(V), Pool(A), Pool(B)]$

3.3.2 Classifiers

1. Logistic Regression: A linear model that predicts the probability of an input belonging to a certain class.
2. K-Nearest Neighbors: A non-parametric algorithm that classifies a data point based on the majority class of its 'k' nearest neighbors in the feature space. I used $k = 5$.
3. Random Forest: An ensemble method that constructs a collection of decision trees at training time. It outputs the class that is the mode of the classes from individual trees. Averaging the predictions of 200 trees reduces variance and combats overfitting. The final prediction $\hat{y}$ for an input x Is:
4. Gradient Boosting: An ensemble technique that builds a strong classifier by sequentially adding weak learners (typically decision trees). Each new tree is trained to correct the errors of the previous one. The model is built in an additive manner:
5. Support Vector Machine: A classifier that finds an optimal hyperplane that maximizes the margin separating data points of different classes.

Table 2: Machine learning Algorithms used for multi-class classification and their mathematical Equations

Classifier	Formula
LogReg	$P(\,y = 1 \mid \boldsymbol{x}\,) = \frac{1}{1 + e^{-(w^{\top}x + b)}}$
KNN	$d(\boldsymbol{p}, \boldsymbol{q}) = \sqrt{\sum_{i=1}^{n} (p_i - q_i)^2}$
RandomF	$\hat{y} = mode\{h_1(\boldsymbol{x}), h_2(\boldsymbol{x}), \dots, h_{200}(\boldsymbol{x})\}$
GBoost	$F_m(\boldsymbol{x}) = F_{m-1}(\boldsymbol{x}) + \gamma_m h_m(\boldsymbol{x})$
SVM	$\min_{\boldsymbol{w}, b, \zeta} \frac{1}{2} \lvert \boldsymbol{w} \rvert^2 + C \sum_{i=1}^{N} \zeta_i$

3.4 Deep Learning Architectures

to achieve better results, I have explored several deep learning models capable of learning spatio-temporal patterns directly from data.

3.4.1 Frame-wise LSTM

I implemented a 2-layer bi-directional Long Short-Term Memory (LSTM) network. The model treats each frame's joint data as a 75-dimensional vector $(25\ joints\ \times\ 3\ coords)$ And processes the sequence of 150 frames to make a classification. In order to compensate for the extreme class imbalance in the dataset, I added regularization techniques, including dropout and a weighted sampler in the data loader. Some of the main key insights of this architecture are:

1. Frame-wise flattening keeps the input tensor small (75 dims) and lets the LSTM learn temporal dynamics directly, avoiding hand-crafted velocity/bone features.
2. Bidirectional layers give the classifier access to past and future context before the final summary vector **h.**

For training the model, I used a data loader with a batch size of 64, an Adam optimizer with 10^{-3} Learning rate and Cross-entropy loss function for 50 epochs.

3.4.2 Temporal 1D-CNN (TCN)

An attempt was made to use a Temporal Convolutional Network[4], which employs dilated 1D convolutions to capture long-range patterns. The initial implementation faced training instability due to exploding activations. Although I tried to mitigate it with normalization and clipping, the model still faltered when the class-balancing weighted sampler was introduced, preventing a final stable result.

3.4.3 Pretrained AGCN

I have evaluated a pretrained 2-Stream Adaptive Graph Convolutional Network (AGCN) [2], a model for skeleton-based action recognition. This is the model that the authors of the BABEL article have used and achieved state-of-the-art results. The “2S” stands for "Two-Stream," which means it uses two input modalities:

1. Joint stream: raw joint positions.
2. Bone stream: bone vectors (differences between joint pairs).

Each stream is processed by an Adaptive Graph Convolutional Network (AGCN), and its outputs are fused for final prediction.

1. Graph Convolution Layer (Spatial Modeling): The skeleton is modeled as a spatial graph $G = (V, E)$, Where nodes are joints and edges are bones [2]. Given input features f_{in}, The graph convolution layer lIs:
$$f_{out}^{(l)} = \sum_{k=1}^{K} W_k^{(l)} f_{in}^{(l)} (A_k + B_k + C_k)$$
2. Temporal Convolution (Temporal Modeling): after Spatial modeling, 1D convolution is applied along the temporal dimension:
$$f_{out}^{(l)} = \text{Conv1D}\left(f_{out}^{(l)}\right)$$
3. Two-Stream Fusion: Each stream has its own AGCN. Their logits are fused (summed or averaged) before the final softmax.
4. Loss Function: For classification, the standard cross-entropy loss is used

3.4.4 Pretrained MotionCLIP

The final model evaluated was the Motion Contrastive Language-Image Pre-training (MotionCLIP)[3] Framework. The core idea is to align motion representations (from a transformer) with CLIP’s text/image embeddings, enabling text-conditioned motion generation, motion-to-text retrieval, and more. It consists of four main components:

1. Motion Encoder: encodes human motion sequences into a latent space.
2. Transformer Decoder: Maps the latent vector to a motion sequence.
3. CLIP Encoder: Pretrained CLIP encodes text or image into a shared semantic space.
4. Latent Alignment: The motion latent (from the decoder) is aligned with the CLIP embedding of text/image via contrastive/cosine losses.

Although the original goal of Motion CLIP is to reconstruct the skeleton coordinates, its aligned CLIP latent space can be used as a motion descriptor for downstream tasks, which is what I have done. I have used its pretrained weights for my Action Recognition task.

Given a motion sequence to the model:

1. It encodes the motion to a CLIP-like feature vector
2. Computes the cosine similarity between the motion feature and all 60 class text embeddings
3. Finally assigns the label with the highest similarity as the prediction.

$$\hat{y} = \arg\max_i \cos\left(f_{\text{motion}}, f_{\text{text}}^i\right)$$

4. Experiments and Results

This section outlines the experimental setup, details the performance of each modeling approach, and provides a comparative analysis of the results.

4.1 Experimental Setup:

4.1.1 Dataset Split

All experiments were conducted on the BABEL-60 action recognition split, which consists of the 60 most frequent action categories in the dataset. The data was divided into standard training, validation, and test sets.

4.1.2 Evaluation Metrics

Model performance was primarily evaluated using Top-1 Accuracy, which measures the proportion of samples where the top predicted class is correct, and Top-5 Accuracy.

4.1.3 Hardware

All model training and evaluation were performed on a local workstation equipped with an NVIDIA RTX 3060 GPU.

4.2 Baseline Performance:

The initial experiments with classic machine learning models indicated the importance of robust feature engineering. A simple SVM trained on “Basic” features failed to learn effectively, achieving an accuracy of only 24.8%, as these features discard critical temporal and relational dynamics.

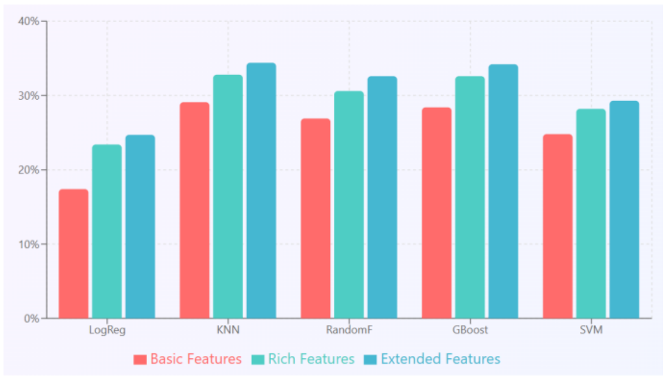

Figure 1: Baseline Models' Performance

As Figure 1 indicates, performance improved significantly with the introduction of richer, engineered features. Table 3 summarizes the Top-1 accuracy of the five baseline classifiers across the different feature sets.

Table 3: performance comparison for Machine learning Algorithms

Model	Basic Features	Rich Features	Extended Features
LogReg	17.40%	23.40%	24.70%
KNN	29.10%	32.80%	34.40%
RandomF	26.90%	30.60%	32.60%
GBoost	28.40%	32.60%	34.20%
SVM	24.80%	28.20%	29.30%

The results show that richer features consistently improved performance across all models. The K-Nearest Neighbors (KNN) model using the "Extended" feature set emerged as the strongest baseline, achieving a Top-1 accuracy of 34.4%.

4.3 Deep Learning Performance:

4.3.1 Frame-wise LSTM

The initial implementation of a bi-directional LSTM indicated promising learning capability but also highlighted a key challenge. As seen in the training curve in Figure 3, the training accuracy steadily climbed to 56.6%, while the validation accuracy peaked at approximately 40.2% around epoch 23 before declining. Correspondingly, the training loss continuously decreased while the validation loss began to rise after epoch 14. This created a significant overfitting gap of ~18%, indicating that the model was memorizing the training data rather than learning generalizable features.

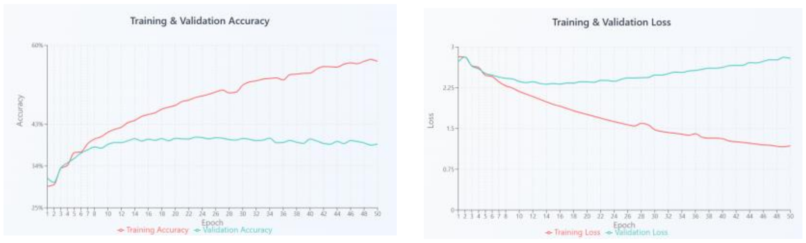

Figure 2: Accuracy and Loss plots for Training and validation Frame-wise LSTM

4.3.2 LSTM (regularized)

In a second iteration, I added stronger regularization, including a higher dropout rate (p=0.5), weight decay, and a weighted sampler to compensate for class imbalance.

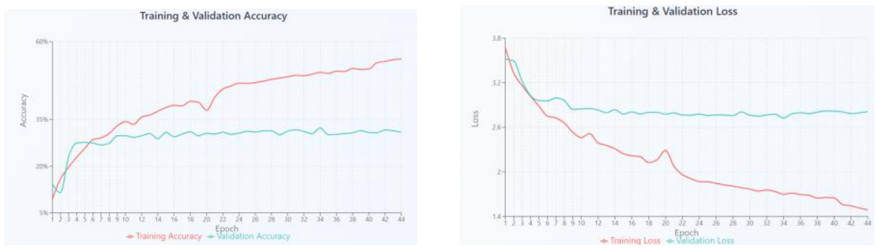

Figure 3: Accuracy and Loss plots for Training and validation Frame-wise LSTM (Improved)

While overfitting was still present, the model achieved a more stable, albeit lower, peak validation accuracy of approximately 31.6%. The final model achieved a Top-1 accuracy of 38.0% and a Top-5 of 68.7% (Figure 4).

4.3.3 Temporal 1D-CNN (TCN)

The TCN model proved challenging to train. Figure 5 shows an initial attempt failed, with exploding activations leading to NaN values in the loss by epoch 7.

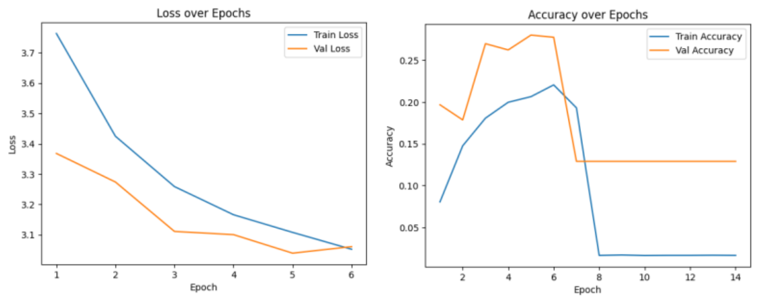

Figure 4: Accuracy and Loss plots for Training and validation TCN

4.3.4 TCN (improved)

A revised pipeline with normalization layers and gradient clipping allowed the model to begin learning. For the first 5 epochs (using standard shuffling), the training and validation loss decreased smoothly. However, upon switching to the class-balancing weighted sampler at epoch 6, the model's performance collapsed. Figure 6 shows a

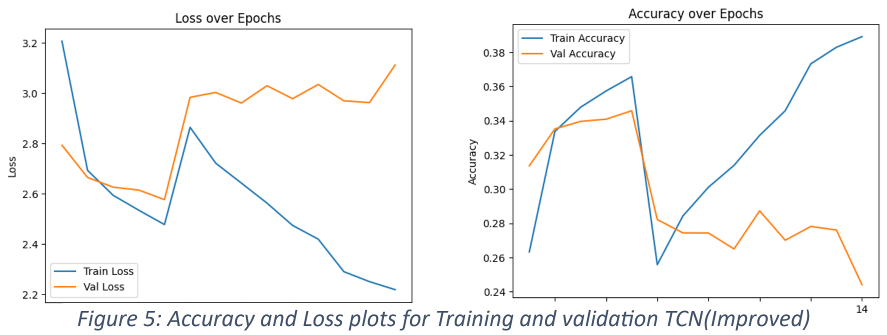

Figure 5: Accuracy and Loss plots for Training and validation TCN(Improved)

sudden spike in validation loss and a sharp drop in accuracy. This indicates the network did not learn robust, class-agnostic features, and the abrupt oversampling of rare classes caused it to immediately over-specialize, destroying its generalization capability.

4.3.5 Pretrained AGCN

Since it was a pretrained model, I only evaluated it in inference mode. After resolving several minor configuration and cross-platform compatibility issues, the model achieved a Top-1 accuracy of 34.0% and a Top-5 accuracy of 68.7% on the validation set.

4.3.6 **Pretrained MotionCLIP:** The MotionCLIP framework also used a pretrained encoder. It achieved the best performance of all evaluated models, reaching a Top-1 accuracy of 45.5% and a Top-5 accuracy of 64.5% on the validation set after the data and visualization pipelines were debugged.

4.4 Summary of Best-Performing Models

As showing in Table 4, these results show that while custom deep learning models outperform classic baselines, large-scale pretrained models like MotionCLIP provide a significant advantage, achieving a ~7.5 percentage point increase in Top-1 accuracy over the best model trained from scratch.

Table 4: Complete Performance Comparison

Model Category	Model Name	Top-1 Accuracy	Top-5 Accuracy
Baseline	SVM	29.3%	-
Baseline	KNN	34.4%	-
Custom	Bi-directional LSTM	38.0%	68.7%
Pretrained	2S-AGCN	34.04%	68.65%
Pretrained	MotionCLIP	45.47%	64.48

5. Discussion and Conclusion

5.1 Discussion of Results

5.1.1 This project systematically evaluated different models, providing several crucial insights into the task of action recognition on the BABEL dataset.
The results of the baseline models underscore the critical importance of feature engineering in classic machine learning. The gap between the performance of the models when they were trained on basic features (mean and standard deviation), rich (velocity, acceleration), and extended features is visible across all of the models. The KNN classifier, achieving 34.4% accuracy, stated a strong baseline for this task.

5.1.2 The custom-trained deep learning models illustrated both the potential and the pitfalls of end-to-end learning on this dataset. The bi-directional LSTM, by achieving 38.0% accuracy, surpassed the best baseline, showing its ability to automatically learn complex temporal dependencies from the raw sequence data. However, its weakness towards overfitting is visible by the large gap between training and validation performance, highlighting a core challenge: without vast amounts of data for all 60 classes, the model struggles to generalize from the BABEL distribution. The training instability of the TCN further reinforces the difficulty of optimizing deep models from scratch under these conditions.

5.1.3 Using the pretrained models indicated the importance of Transfer Learning. While the pretrained AGCN performed comparably to the baseline, the MotionCLIP framework achieved a standout accuracy of 45.5%. This performance is the result of its disentanglement learning process[3], which helped the model to generalize much better by learning a sparse representation of BABEL.

5.2 Limitations

Firstly, due to technical challenges with training stability and runtime errors, a full evaluation of the custom TCN and ST-GCN architectures was not completed. Resolving these issues would provide a more complete comparison of different deep learning architectures. Secondly, the pretrained models (AGCN and MotionCLIP) were evaluated in a zero-shot manner on the test set, without any fine-tuning.

5.3 Future Work

5.3.1 Fine-tuning

The next step is to fine-tune the high-performing MotionCLIP encoder on the AMASS.[5] Dataset in an unsupervised manner.

5.3.2 Semi-supervised

Using the BABEL dataset as the smaller labeled dataset to train a model, and then creating pseudo-labels based on the initial model[6] . This way, we can exploit more features and create a more generalized model.

5.3.3 Generative models

using diffusion-based architecture to create similar data and augment it into the training process for a more generalized model.

5.4 Conclusion

The goal of this project was to systematically evaluate different approaches towards human action recognition using the BABEL dataset. By progressing from classic machine learning models to custom-trained deep learning models and finally to large-scale pretrained architectures, I demonstrated a clear hierarchy of performance. While hand-crafted features provided a solid baseline and custom LSTMs learned effective temporal patterns, both were ultimately limited by the dataset's complexity and class imbalance. The state-of-the-art performance was achieved by the pretrained MotionCLIP framework, highlighting that the most effective strategy for tackling challenging, large-scale action recognition tasks is to leverage the powerful, generalized knowledge encapsulated in large, pretrained models.

References

[1] A. R. Punnakkal, A. Chandrasekaran, N. Athanasiou, A. Quirós-Ramírez, and M. J. Black, “BABEL: Bodies, Action and Behavior with English Labels,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2021. doi: 10.1109/CVPR46437.2021.00078.

[2] L. Shi, Y. Zhang, J. Cheng, and H. Lu, “Two-stream adaptive graph convolutional networks for skeleton-based action recognition,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019. doi: 10.1109/CVPR.2019.01230.

[3] G. Tevet, B. Gordon, A. Hertz, A. H. Bermano, and D. Cohen-Or, “MotionCLIP: Exposing Human Motion Generation to CLIP Space,” in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2022. doi: 10.1007/978-3-031-20047-2_21.

[4] S. Yan, Y. Xiong, and D. Lin, “Spatial temporal graph convolutional networks for skeleton-based action recognition,” in *32nd AAAI Conference on Artificial Intelligence, AAAI 2018*, 2018.

[5] N. Mahmood, N. Ghorbani, N. F. Troje, G. Pons-Moll, and M. Black, “AMASS: Archive of motion capture as surface shapes,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2019. doi: 10.1109/ICCV.2019.00554.

[6] Y. Xu *et al.*, “Cross-Model Pseudo-Labeling for Semi-Supervised Action Recognition,” in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2022. doi: 10.1109/CVPR52688.2022.00297.