Eidgenössische Technische Hochschule Zürich
Swiss Federal Institute of Technology Zurich

Distributed Computing

# Efficient Real-Time Egocentric Action Recognition

Master's Thesis


Marco Calzavara

mcalzavara@student.ethz.ch

Distributed Computing Group
Computer Engineering and Networks Laboratory
ETH Zürich


**Supervisors:**
Ard Kastrati, Matteo Macchini, Dushan Vasilevski
Prof. Dr. Roger Wattenhofer

December 4, 2024

# Acknowledgements

I would like to express my sincere gratitude to my thesis supervisors, Ard Kastrati, Matteo Macchini, Dushan Vasilevski, and Professor Roger Wattenhofer, for their help and guidance throughout my master's thesis journey.

I am also deeply thankful to the Distributed Computing Group and the Computer Engineering and Networks Laboratory at ETH Zurich for providing the computing resources that were crucial for the success of this work.

Finally, I extend my heartfelt thanks and appreciation to my family and friends, whose constant encouragement and support have been a source of motivation throughout this process.

# Abstract

Egocentric Action Recognition (EAR) has emerged in recent years as a new area of research due to the increasing use of wearable devices such as AR glasses. These devices face significant challenges in real-time action recognition due to limited computational resources and the need to preserve battery life for an optimal user experience. This project focuses on optimizing the trade-off between prediction performance, inference speed, and CPU usage for action recognition systems. By leveraging sequences of RGB frames and 3D hand pose keypoints, we explore how reducing the sampling frequency of these modalities affects performance, speed, and resource usage. Our approach aims to enable efficient real-time action recognition on AR glasses without compromising user experience and battery life.

# Contents

Chapter 1

# Introduction

Since the advent of head-mounted devices, researchers and industries have shown increasing interest in using videos captured by wearable cameras to extract valuable information. Among these, AR glasses such as Magic Leap 2 stand out as devices that integrate digital information into the user's view of the physical world. Leveraging the data captured by these glasses can unlock numerous applications, ranging from task assistance to personalized recommendations and contextual awareness.

Given the inherent differences between the exocentric and egocentric points of view, recognizing actions from egocentric data has evolved into a distinct research field known as Egocentric Action Recognition (EAR). EAR enables devices to understand the wearer's actions, enhancing the user experience by providing downstream modules, such as recommendation systems or digital assistants, with insights about the wearer's behavior and context.

However, EAR presents unique challenges due to the variability of egocentric viewpoints, occlusions, and limited computational resources on wearable devices. For AR glasses, computational efficiency, measured in terms of inference speed and battery life, is paramount. These devices must deliver real-time performance while maintaining long battery life, ensuring practical usability.

This project addresses these challenges by focusing on real-time, accurate, and efficient action recognition tailored for AR glasses. Head-mounted devices often lack access to dedicated hardware acceleration and must instead rely on limited, shared computational resources. This constraint makes achieving real-time action recognition on the CPU particularly challenging. Vision models, which are essential for extracting valuable cues from the RGB modality, are especially demanding in terms of computational resources. They rarely achieve real-time performance on a CPU, which makes deploying action recognition systems on AR glasses even more challenging. Furthermore, minimizing CPU usage is critical to preserving battery life, a key factor for ensuring the usability of AR glasses in real-world applications.

Our approach leverages sequences of RGB frames and 3D hand pose key-

points to predict the current action, aiming to optimize both inference speed and resource efficiency. We analyze the impact of reducing the sampling frequency of these modalities on performance, speed, and CPU usage, exploring the trade-offs between these competing requirements through design choices and experimental results.

Chapter 2

# Background

Most research on EAR builds upon the idea that objects are the most important mode for accurate action recognition. In the EAR field, objects indeed posses properties that make them extremely valuable cues: first, identifying *active objects* —those being acted upon— is often key to distinguishing actions; second, manipulated objects typically lie within the *observable space*. However, focusing solely on object detection in visual understanding may overlook valuable environmental and interaction-based information.

While object appearance plays a significant role in egocentric action recognition, it is not the only important cue. Researchers have explored alternative egocentric data modalities, such as head pose, hand pose, and gaze direction, which provide unique insights from the egocentric viewpoint. Existing approaches in the field of EAR are discussed further in Section 2.1, while Section 2.1.3 offers an overview of egocentric datasets. Lastly, Section 2.2 describes the vision backbone models employed in this project.

## 2.1 Egocentric Action Recognition

The literature on Egocentric Action Recognition (EAR) can be categorized in various ways. For instance, [1] classifies approaches into three primary categories: Object-based, Motion-based, and Hybrid methods, with a residual category labeled as *others* for works that do not fit into these groups. However, for the purposes of this work, a more suitable taxonomy focuses on two classes: Single-modality systems and Multi-modal systems.

In this work, we define a multi-modal system as one that incorporates distinct data modalities provided explicitly as input, where each modality represents a unique and independent source of information. These modalities are typically processed separately and later fused to form a richer representation. Examples include systems that leverage RGB frames alongside optical flow, depth data, or gaze direction, with each provided as an input to the model and originating from an external model independent of the main action recognition task.

By contrast, single-modality systems rely on a single data source, such as RGB frames, and may include intermediate features generated within the model pipeline (e.g., segmentation masks, bounding boxes, or motion cues). Such features are not considered separate modalities because they are derived from the primary input and computed as part of an end-to-end processing pipeline.

For example, if 3D hand pose data is pre-computed by a separate hand-tracking system that is not jointly trained with the action recognition model, it is treated as an independent data modality. In contrast, if the 3D hand pose is generated by a model that is also trained on the recognition loss, it is not regarded as a separate modality.

### 2.1.1 Single-Modality Models

Several studies in the field of Egocentric Action Recognition have explored using a single data source for prediction. For example, [2, 3] adopt a Bag-of-Objects approach that relies exclusively on the RGB modality for object detection. Both works differentiate between objects being actively manipulated by the subject and those that are not. Similarly, [4] proposes a method that generates frame-level feature vectors from object detections obtained using YOLO [5].

While object detection has demonstrated promising results in action recognition tasks, alternative methods focus on feature extraction directly from RGB data. For instance, [6] combines a 2D CNN for single-frame feature extraction, a 3D CNN for integrating spatial features across two consecutive frames, and a convLSTM [7] for temporal aggregation. Inspired by optical flow-based methods, [6] also investigates the use of frame differences as input instead of raw frames. This approach is shown to achieve competitive performance compared to multi-modal models. On the other hand, [8] operates directly on optical flow, processing motion information stacked over 4 seconds of video. While this model exclusively uses optical flow, many successful multi-modal architectures combine optical flow features with appearance-based features (see Section 2.1.2).

Though numerous works on single-modality EAR have demonstrated that the RGB modality alone can yield competitive results, an interesting research question is whether actions can be predicted solely from hand pose information, without relying on appearance or the identity of active objects. For instance, [9] utilizes ground truth hand segmentation to mask out all non-hand background, feeding the resulting frames into a CNN for action recognition. Similarly, [10] employs a Spatio-Temporal Graph Convolutional Network to process sequences of hand skeleton data points. This architecture draws inspiration from [11], which successfully applies a graph neural network for exocentric action recognition using full skeleton data. Although these models do not produce perfect results, they demonstrate a significant correlation between hands and egocentric actions, supporting the use of hand-related data for EAR.

### 2.1.2 Multi-Modal Models

The primary limitation of single-modality systems is their reliance on a single data source to learn complex properties such as object identity, state changes, hand-object interactions, and motion. As a result, they may fail to capture critical cues that are intrinsic to egocentric action recognition. Multi-modal models address this limitation by incorporating additional data modalities as input.

Figure 2.1 illustrates a typical architecture of a multi-modal system. Each modality is processed through a dedicated feature extractor, and the resulting feature vectors are fused to make a final prediction.

A widely used multi-modal system is the two-stream architecture, initially proposed in [12] for third-person action recognition. This model consists of two streams, one for processing RGB frames, and the other for processing stacked optical flow. This design allows the RGB stream to capture single-frame appearance features, while the optical flow stream learns motion patterns across a sequence of frames. [13, 14, 15, 16, 17, 18] have adapted the two-stream architecture for egocentric action recognition. For instance, [14] proposes a model that jointly performs gaze estimation and egocentric action recognition. This two-stream model produces a gaze map as an intermediate output, which is then used as an attention map to refine the visual features. While ground truth gaze locations are required during training, they are estimated on-the-fly during inference using the fused features. Similarly, [15, 16] utilize ground truth gaze measurements to train gaze estimation modules, which are then employed to generate attention maps. Furthermore, [18] demonstrates that adding an optical flow branch to the model proposed in [6] enhances accuracy on the egocentric action recognition task by approximately 14%, confirming that the RGB stream benefits from not having to learn motion.

While most research has focused on combining optical flow with the RGB frames, some studies investigate different data modalities. For example, [19, 20] use Inertial Measurement Unit (IMU) data in conjunction with RGB frames for first-person action recognition. Additionally, [21] proposes a model that processes RGB frames and 3D hand poses in chunks. In this network, a micro-segment encoder slides over the chunks to generate feature vectors, which are then processed by a temporal transformer module. Although these approaches are less explored compared to classical two-stream architectures, they have the advantage of avoiding optical flow computation, which is generally resource-intensive. In contrast, IMU data and 3D hand pose are readily available or can be obtained with minimal resource requirements.

A natural extension of two-modality architectures is the inclusion of additional input modalities. For instance, [22] employs a three-stream architecture comprising a spatial branch for RGB frames, a motion branch for optical flow, and an object branch for object-based features. The study demonstrates that

combining all three streams outperforms using any single stream, highlighting the complementary nature of these modalities. Similarly, [23] integrates RGB frames, optical flow, and stacked egocentric cues, namely hand masks, head motion, and saliency maps. In another approach, [24] incorporates depth maps alongside RGB and optical flow inputs, while [25] proposes an architecture that takes RGB, depth, and gaze as input.

Given the modularity of multi-modal architectures, it is theoretically possible to incorporate an arbitrary number of data modalities. However, the primary limiting factor is the availability of these modalities, as they are not always supported by commonly used wearable cameras or AR glasses. Furthermore, adding more data modalities does not always result in improved performance, as demonstrated by [26]. The authors show that the performance of a model using only RGB is comparable to, if not better than, models that additionally utilize hand pose, hand pose with eye gaze, or all three modalities combined with head pose.

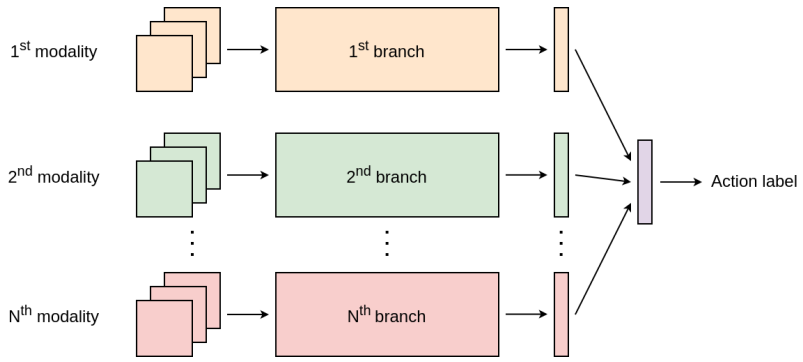


Figure 2.1: General architecture of a multi-modal system. Separate branches process individual modalities, and the extracted features are fused for final prediction.

### 2.1.3 Egocentric Datasets

Datasets play a crucial role in driving advancements in machine learning. Since the emergence of egocentric action recognition as a research field, several datasets have been introduced, significantly contributing to the improvement of prediction models. Table 2.1 lists some of the most relevant egocentric datasets and their properties, while Table 2.2 details the data modalities available for these datasets. The following section provides an overview of the datasets listed in Tables 2.1 and 2.2, highlighting their key features.

**ADL**. The Activity of Daily Living (ADL) dataset [2] contains over 10 hours of egocentric footage showcasing activities performed by 20 subjects in home environments. It provides action labels along with object bounding boxes, further annotated by their state (active or passive).

**EGTEA Gaze+**. The Extended Georgia Tech Egocentric Activity (EGTEA Gaze+) dataset [14, 27] features 29 hours of cooking activities performed by 32 subjects. It includes high-resolution RGB videos, audio, gaze annotations, and hand masks for approximately 14000 frames.

**Charades-Ego**. The Charades-Ego dataset [28, 29] offers 43,594 labeled actions with both egocentric and exocentric views of indoor activities, covering a wide variety of subjects and environments.

**FPHA**. The First-Person Hand Action (FPHA) dataset [30] provides RGB-D data, 3D hand poses, and 6D object poses and meshes for four objects. The recorded actions span kitchen, office, and social settings.

**EPIC-KITCHENS-55**. EPIC-KITCHENS-55 [31] is a large-scale egocentric dataset of kitchen activity recordings.

**EPIC-KITCHENS-100**. The EPIC-KITCHENS-100 dataset [32] builds on [31] by extending it and providing additional annotations, such as masks for hands and objects, along with hand-object interaction detections.

**H2O**. The H2O dataset [33] includes multi-view RGB-D images, ground-truth 3D hand poses for both hands, camera poses, and 6D object poses and meshes.

**BON**. The BON dataset [34] focuses solely on the RGB modality and is one of the few egocentric datasets featuring office activity recordings.

**Assembly101 (ego)**. The Assembly101 egocentric dataset [35] showcases videos of individuals assembling and disassembling toy vehicles in an industrial-like setting. It provides both egocentric and exocentric perspectives.

**HOI4D**. HOI4D [36] is an egocentric dataset of indoor activities with extensive annotations, including hand and object poses.

**Meccano**. The Meccano dataset [37, 25] captures egocentric recordings in an industrial-like domain, where participants build a toy motorbike.

**HoloAssist**. HoloAssist [26] is an egocentric dataset offering seven raw sensor modalities: RGB, depth, head pose, 3D hand pose, eye gaze, audio, and IMU data.

## 2.2 Vision Backbone Models

Images are a crucial source of information for egocentric action recognition. However, extracting meaningful information from raw RGB frames poses significant challenges, often requiring the use of complex vision models. This section explores the backbone models utilized in this work.

| Dataset | Segments | Frames | Action Cls | Subj | Obj Cls | Setting |
|---|---|---|---|---|---|---|
| ADL [2] | 436 | 1M | 32 | 20 | 42 | Indoor |
| EGTEA Gaze+ [27, 14] | 10325 | ∼ 2.5M | 106 | 32 | 53 | Kitchen |
| Charades-Ego [28, 29] | 43594 | ∼ 2.9M | 157 | 102 | 36 | Indoor |
| FPHA [30] | 1175 | 100k | 45 | 6 | 26 | Multiple [1] |
| EPIC-KITCHENS-55 [31] | 39564 | 11.5M | 101 [2] | 32 | 46 [2] | Kitchen |
| EPIC-KITCHENS-100 [32] | 89,977 | 20M | 239 [2] | 37 | 85 [2] | Kitchen |
| H2O [33] | 933 | 100k | 36 | 4 | 8 | Indoor |
| BON [38, 34] | 2639 | ∼ 400k | 18 | 25 | N/A | Office |
| Assembly101 (ego) [35] | 330k [3] | 36M | 1380 [3] | 53 | 90 [3] | Industrial |
| HOI4D [36] | 4000 | 2.4M | 54 | 9 | 16 | Indoor |
| Meccano [37, 25] | 8857 | N/A | 61 | 20 | 20 | Industrial |
| HoloAssist [26] | N/A | ∼ 18M | 414 [4] | ∼ 100 | 16 | Assistive task |

[1] Kitchen, Office, Social Interactions.
[2] Only action classes with more than 50 samples in the training set are included.
[3] Values correspond to fine-grained actions.
[4] Coarse-grained actions.

Table 2.1: Overview of prominent egocentric datasets for action recognition.

| Dataset | Image Type | Depth | Audio | Gaze | Objects | Hands |
|---|---|---|---|---|---|---|
| ADL [2] | RGB | × | × | × | BBs | × |
| EGTEA Gaze+ [27, 14] | RGB | × | ✓ | ✓ | × | Masks |
| Charades-Ego [28, 29] | RGB | × | × | × | × | × |
| FPHA [30] | RGB | ✓ | × | × | Obj Poses | 3D Poses |
| EPIC-KITCHENS-55 [31] | RGB | × | ✓ | × | BBs | × |
| EPIC-KITCHENS-100 [32] | RGB | × | ✓ | × | Masks | Masks |
| H2O [33] | RGB | ✓ | × | × | Obj Poses | 3D Poses |
| BON [38, 34] | RGB | × | × | × | × | × |
| Assembly101 (ego) [35] | Gray | × | × | × | × | 3D Poses |
| HOI4D [36] | RGB | ✓ | × | × | Obj Poses | 3D Poses |
| Meccano [37, 25] | RGB | ✓ | × | ✓ | BBs | BBs |
| HoloAssist [26] | RGB | ✓ | ✓ | ✓ | × | 3D Poses |

Table 2.2: Overview of data modalities provided by egocentric datasets.

### 2.2.1 CNNs

For years, Convolutional Neural Networks (CNNs) have been the state of the art in computer vision, driving significant advancements across a wide range of tasks, including object detection, image classification, semantic segmentation, and action recognition. Their strength lies in their ability to hierarchically learn both low-level features, such as edges, and high-level features, like objects, making them effective for many visual tasks.

#### RegNet

While years of research have deepened our understanding of network design, many CNNs networks remain highly tuned for specific settings. RegNet, introduced by [39], addresses this limitation by offering a family of simple and effective networks that generalize well across various settings. The authors propose an approach to network design that begins with a broad design space and progressively applies constraints. Each constraint is retained only if it does not degrade the error distribution compared to the original space, as evaluated using models with similar computational cost (in FLOPs). The resulting RegNet design space enables the sampling of high-performing networks across different computational regimes, often outperforming state-of-the-art CNNs.

### 2.2.2 ViTs

The Transformer architecture [40] has established itself as the standard for natural language processing tasks due to its efficiency and scalability. Its application to vision tasks gained widespread popularity with the introduction of the Vision Transformer (ViT) [41]. ViT follows the same architectural principles as [40], obtaining the sequence of tokens by dividing the input image into patches, flattening each patch, and linearly projecting the resulting vector to obtain an embedding of the desired dimensionality.

Despite showing promising results in vision tasks, state-of-the-art ViT architectures often struggle to generalize well when trained on limited data, particularly in comparison to CNNs, which benefit from inductive biases. As a result, ViTs typically require extensive pre-training on large datasets to achieve competitive results. Moreover, ViTs are generally not suited for real-time applications on resource-constrained devices due to their computational complexity.

#### DeiT

DeiT [42] addresses the dependency of transformers on large pre-training datasets. The proposed strategy consists of using several data augmentation techniques

which effectively increase the size of the training dataset across epochs without altering the number of intra-epoch batches. Based on ablation studies, the authors identify and utilize several techniques, including Rand-Augment [43], Mixup [44], CutMix [45], random erasing [46], repeated augmentation [47], and stochastic depth [48]. These augmentations, detailed in Section 4.3, result in performance improvements over ViT-B/16 and the larger ViT-L/16 [41] when trained on ImageNet [49] without external pre-training data.

DeiT further increases performance through distillation [50], which is used to transfer the knowledge of a RegNet teacher model to the Vision Transformer.

### LeViT

To address the inference speed issue, [51] introduces a family of hybrid neural networks that optimize the speed-accuracy trade-off. While achieving inference speeds beyond 30 fps may seem excessive, architectures with higher throughput offer better battery efficiency, a critical factor in many real-world applications. LeViT models feature more parameters than DeiT models for equivalent accuracy but achieve higher throughput by incorporating convolutional layers and attention-shrinking modules. Similar to DeiT, [51] employs distillation [50] to enhance classification performance.

Chapter 3

# Model Architectures

In this chapter, we describe the architectures of the models used throughout the project. The chapter is divided into two sections: Section 3.1 focuses on models that process individual frames, while Section 3.3 covers models designed to operate on sequences.

## 3.1 Single-frame models

### 3.1.1 LeViT

For the RGB data modality, we employ the LeViT-256 model [51] as a feature extractor. Additionally, we train and evaluate this model on the task of predicting actions from single RGB frames. The architecture of LeViT is inspired by the Vision Transformer [41], but incorporates some design changes for better inference speed. First, the patch flattening operation is replaced by four layers of 3x3 convolutions. This modification is justified by an exploratory analysis showing that placing a convolutional network before a transformer improves performance under speed constraints. Second, LeViT reduces resolution across stages using a shrink-attention layer, which further optimizes its efficiency.

Unlike ViT [41] and DeiT [42], LeViT does not produce an embedding for each input token alongside a class token. Instead, it applies an average pooling operation after the final transformer layer to obtain a single feature vector, which can be used as input for classification.

Inspired by prior works [50, 42, 51], we place two classification heads on top of the final average pooling layer, as shown in Figure 3.1. During inference, the logits produced by the two classification heads are averaged, and the resulting vector is used to compute the probabilities. We employ RegNetY-12GF [39] as the teacher model.

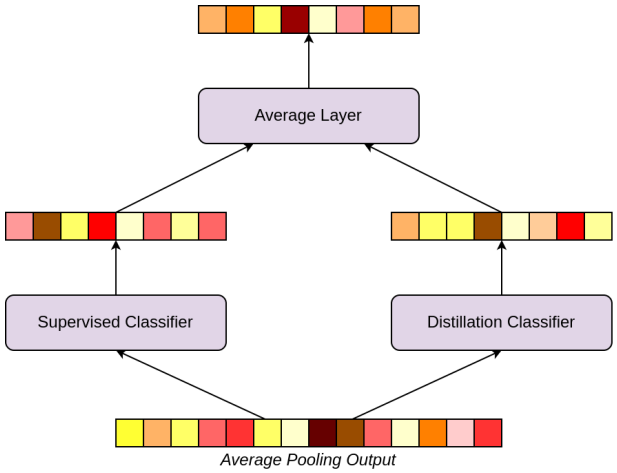


Figure 3.1: Architecture of the classification module of LeViT.

### 3.1.2 HP-MLP

For the hand pose modality, we employ a simple yet effective shallow network, whose architecture is shown in Figure 3.2. The input to the network consists of the flattened, normalized hand pose data concatenated with the flattened rotation matrices of the two hands. This simple network performs well despite not explicitly accounting for the intrinsic graph structure of the hands. This could be due to a variety of reasons:

- Hand Pose data consists of a small number (42) of data points, each with only 3 features. Using more complex or deeper architectures might not improve performance given such a low-dimensional input.
- Although the input has intrinsic connectivity, this connectivity does not vary across data points. The absence of variance in the adjacency matrix could explain why connectivity-aware models do not outperform the simple network.
- The hand pose normalization, described in Chapter 4, significantly reduces input variance, making it easier for a shallow network to converge.
- A shallow network, batch normalization [52], and dropout [53] effectively reduce the risk of overfitting.

## 3.2 FusionNet: LeViT + HP-MLP

LeViT and the hand pose model each provide solutions to the unimodal single-frame prediction task. To explore the potential of combining these models, we

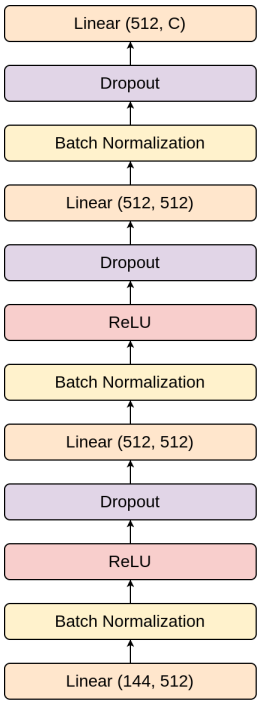


Figure 3.2: Architecture of the hand pose model.

introduce a fusion architecture, as illustrated in Figure 3.3. The goal of this architecture is to leverage the complementary information from the two different modalities to improve overall performance. Prior to fusion, we apply two projection layers, each followed by a normalization step, which makes the feature representations from both models more comparable, ultimately enabling a more effective fusion process.

## 3.3 Sequence models

Operating over sequences is crucial for accurate action recognition, as context from multiple consecutive time steps helps detect motion. In this project, we use single-frame models for feature extraction and process the resulting sequence of features using either a Temporal Convolutional Network (TCN) [54, 55] or a sequence model inspired by [56].

### 3.3.1 TCN

Traditional Convolutional Neural Networks (CNNs) face challenges when applied to sequences of feature vectors. Specifically, the receptive field grows slowly when using small kernels and strides. This slow growth necessitates either larger kernels

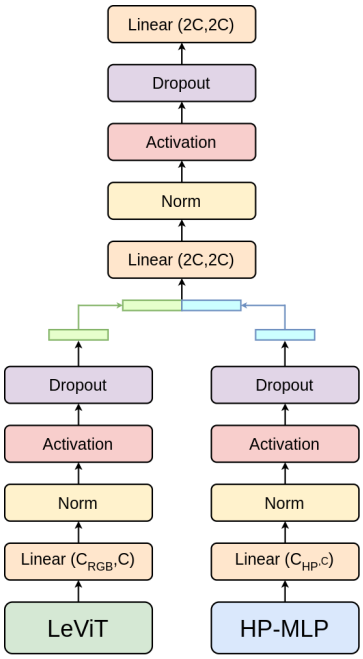


Figure 3.3: Architecture of FusionNet model.

or deeper networks to process longer sequences, which can lead to increased risk of overfitting and slower inference speed. Temporal Convolutional Networks (TCNs) [54, 55] address this limitation through the use of dilated convolutions [57]. Dilated convolutions allow the receptive field to grow exponentially even with small kernel sizes, enabling the network to efficiently process long sequences while maintaining a small number of parameters.

Figure 3.4 illustrates the architecture of the Temporal Convolutional Network (TCN). Each residual block in the TCN contains two dilated convolutions, each followed by a normalization layer, an activation function, and a dropout layer [53]. We employ Layer Normalization [58] and ReLU as the normalization and activation functions, respectively. The residuals from each block are projected to match the output dimension and summed with the output of the final block to produce the network's output.

In the original TCN implementation, the convolutions are causal, meaning the output at position t depends only on positions less than or equal to t. However, we modify this by using acausal convolutions with padding, as our objective is to compute the output at the final position, which cannot depend on future inputs.

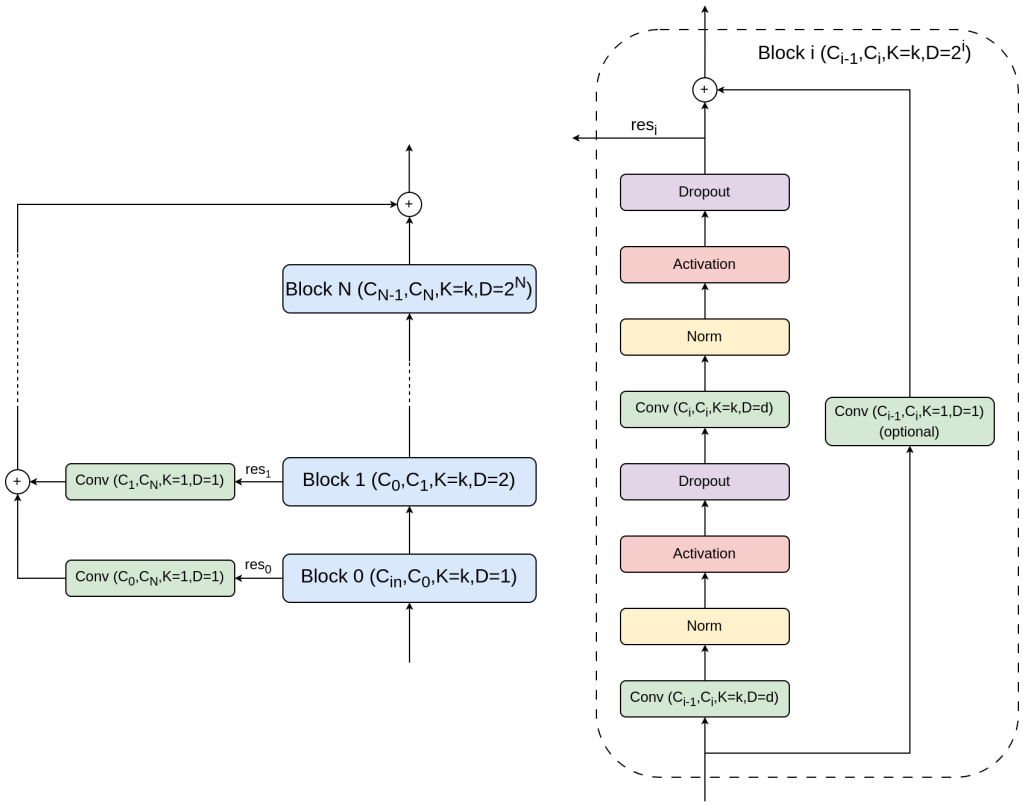


Figure 3.4: Architecture of the Temporal Convolutional Network.

### 3.3.2 Temporal MLP

The primary challenge with TCNs lies in their numerous hyperparameters, which are difficult to tune. Additionally, they require careful adjustment to ensure that the receptive field at the final layer's last position covers the entire sequence length. To address these issues, we develop and test an alternative sequence model, which we call Temporal MLP (TMLP).

Figure 3.5 shows the architecture of the TMLP model, inspired by [56]. Each block begins by projecting the input to the block dimension, if necessary, and adding projected positional embeddings. The result is then passed through two residual modules. Each module consists of a normalization operation, an activation function, and either a convolution or a linear layer in between.

The convolution facilitates mixing along the temporal dimension, as the kernel with filter size 1 slides along the channel dimension, effectively processing vectors whose size corresponds to the sequence length. In contrast, the linear layer operates across the temporal dimension, enabling mixing of the channels within each time step. This design allows the receptive field to cover the entire sequence after just one block.

The main advantage of the Temporal MLP over the Temporal Convolutional Network is its lower number of hyperparameters, which simplifies testing. In practice, we use SiLU [59] as the activation function and Layer Normalization [58] as the normalization layer.

### 3.3.3 Multimodal Temporal MLP

The architecture depicted in Figure 3.6, which we call MM-MLP, represents a multimodal fusion framework designed to combine RGB and hand pose (HP) features from sequences. Each modality undergoes feature extraction and temporal processing through dedicated modules: an RGB Feature Extractor and Temporal MLP for RGB features, and an HP Feature Extractor paired with a Temporal MLP for HP features. The outputs from both streams, corresponding to the final time steps, are concatenated and processed through a series of layers, including a fully connected linear layer for feature merging, followed by normalization, activation, dropout, and a final linear projection to produce the output logits. This design offers significant flexibility: it supports the integration of any number of modalities, each with its own dedicated stream; its modular structure allows the use of various feature extractors and sequence models - for instance, a TCN could be used in place of the Temporal MLP; finally, the separation of modalities enables the use of different context lengths and sampling frequencies.

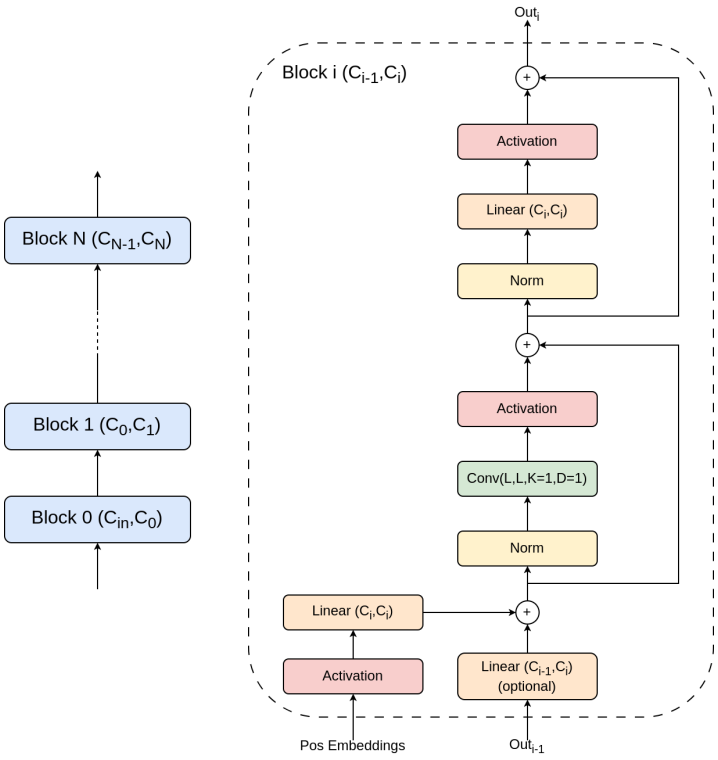


Figure 3.5: Architecture of the Temporal MLP.

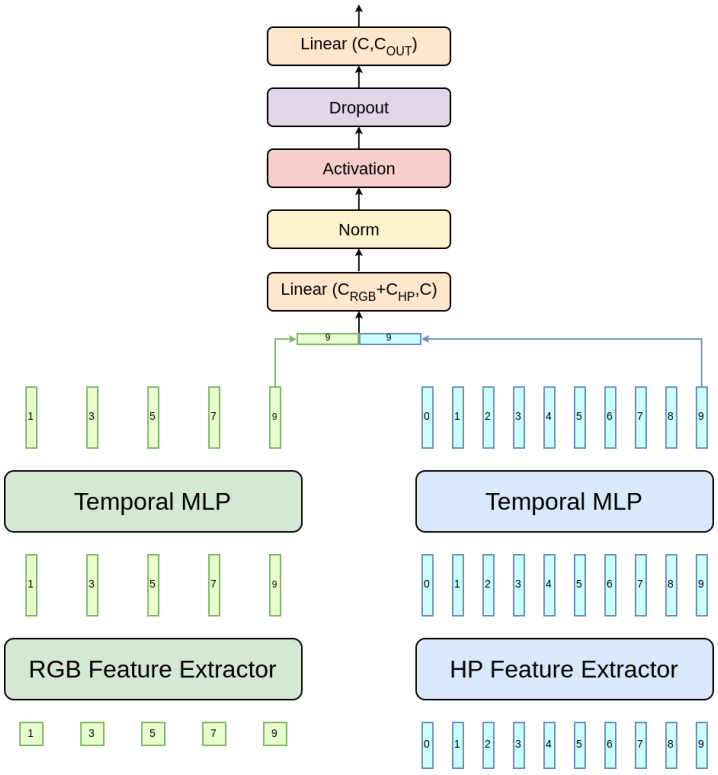


Figure 3.6: Architecture of the Multimodal Temporal MLP.

Chapter 4

# Experimental Setup

## 4.1 Dataset

We utilize the H2O dataset [33] to conduct our experiments. The dataset's statistics can be visualized in Appendix A. While the H2O dataset is designed for predicting the action performed over a video segment, it can also be used for frame-level prediction by assigning the segment's action label to each frame within the segment.

In this section, we discuss how we generate sequences of data points from the segments. Consider a segment with two data modalities, as shown in Figure 4.1, and assume a sequence length of 10 time steps for both modalities. This corresponds to one-third of a second, given that the H2O dataset was recorded at 30 fps. Our goal is to analyze how different combinations of sampling frequencies for the two modalities affect performance. In this example, we set a frequency of 10 Hz for the RGB modality and 30 Hz for the hand modality. Figure 4.2 illustrates the sequence generation process for training. Starting from the first time step, we slide a window of width equal to the sequence length along the segment, creating two sequences that end at the current frame. If the window does not contain enough time steps, we duplicate the first time step to fill the missing spots. Then, we sample the time steps from the sequence starting from the end, which ensures that the last time step is always included.

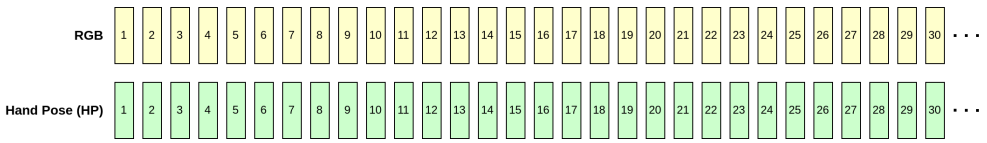


Figure 4.1

In a real-world scenario, frames and hand pose data would be streamed online. If we were to always consider aligned time steps for both modalities, as done during training, we would lose the advantage of using a smaller frequency for the computationally expensive RGB stream, since the feature extractor would need

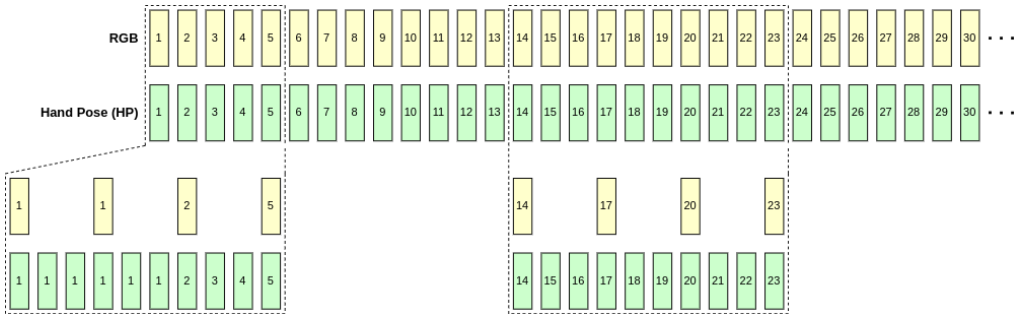


Figure 4.2

to process every incoming frame. Instead, we extract features only at the desired frequencies, as illustrated in Figure 4.3, where the sampled time steps are shown in a darker shade. Figure 4.4 demonstrates two examples of sequences used for prediction at different time steps. At time step 8, the sampled frames up to time step 7 are used for the RGB modality, as frame 8 is not sampled. To maintain the required sequence length, the first frame is duplicated. At time step 24, we utilize the four most recently sampled frames from the RGB stream and the last ten 3D hand poses from the hand pose stream.

The described approach involves maintaining a fixed-size buffer for each modality, initially filled with duplicates of the features from the first time step. A new sample is added to the buffer whenever a modality-specific, frequency-dependent timer expires, while the least recent sample is removed to ensure the buffer remains within the fixed size. The strategy can also be utilized if modalities are not perfectly aligned - for instance, when the 3D hand pose is computed from the RGB frames, and becomes available a few milliseconds after the image, as illustrated in Figure 4.5.

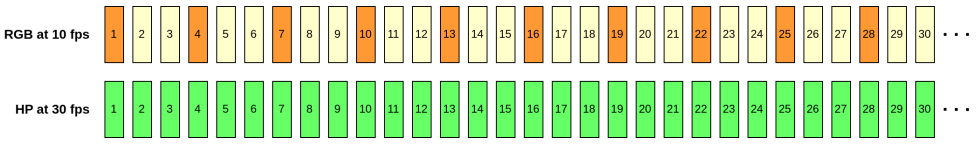


Figure 4.3

## 4.2 Preprocessing

### 4.2.1 RGB frames

Before being processed by the RGB feature extractor, the RGB frames undergo cropping and normalization. Cropping involves determining the minimal bounding box around the hand keypoints projected onto the image plane. This bound-

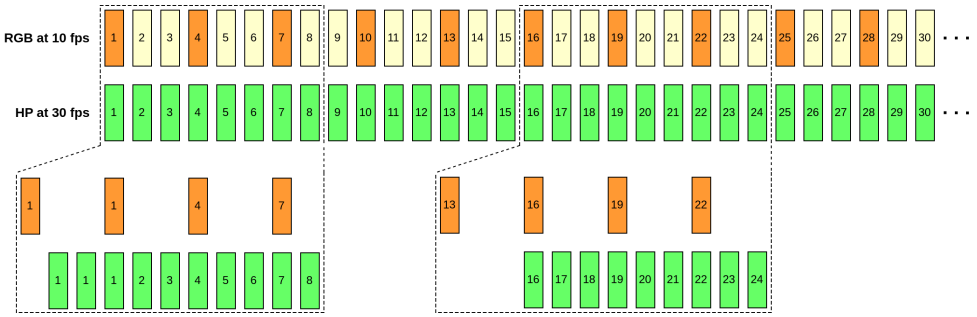


Figure 4.4

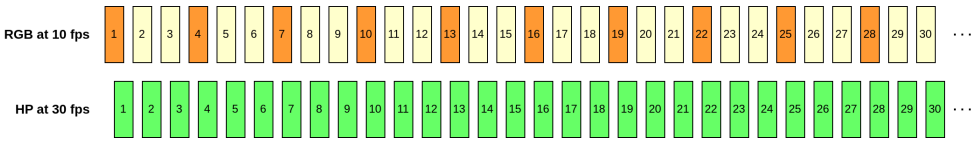


Figure 4.5

ing box is then uniformly extended in all directions to meet the size requirements of the feature extractor. The detailed cropping procedure is illustrated in Figure 4.6.

Normalization adjusts the pixel intensity values of the cropped frames to align with a Normal distribution. Specifically, for each channel (Red, Green, and Blue), the pixel values are transformed using the formula:

$$\text{output}[\text{channel}] = \frac{\text{input}[\text{channel}] - \text{mean}[\text{channel}]}{\text{std}[\text{channel}]}.$$

Here, mean[channel] and std[channel] are the channel-wise mean and standard deviation values computed from the ImageNet dataset [49].

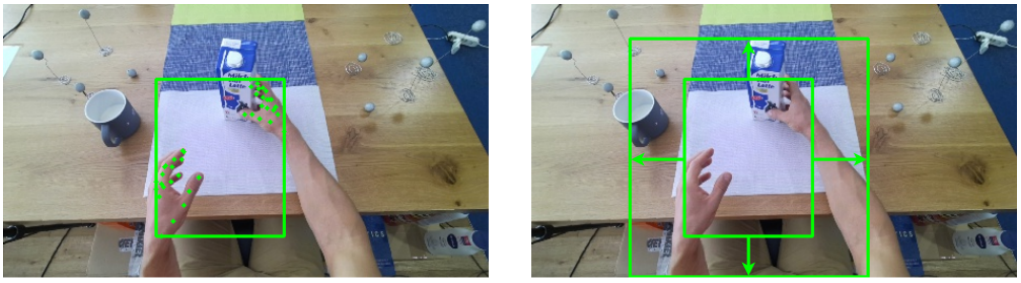

Figure 4.6: Example of cropping based on projected 2D hand keypoints. The minimal bounding box is expanded to meet the 224x224 input size required by the LeViT model.

### 4.2.2 3D Hand Pose

The 3D hand keypoints undergo a normalization process consisting of three steps:

1. **Translation to the Origin**: Both hands are translated such that the wrists are moved to the origin of the coordinate system. The original positions of the wrists are saved for later use. This translation ensures that the wrist joint serves as a reference point, which is consistent across samples.
2. **Edge Length Modification**: The lengths of the edges between keypoints are adjusted so that corresponding edges from different hand samples have the same length. This step helps to remove variations in hand size across samples.
3. **Rotation Alignment**: The hands are rotated so that the following properties are satisfied:
   - The vector from the wrist to the middle metacarpal (the keypoint representing the middle finger's base) is aligned with the negative y-axis.
   - The vector from the fifth metacarpal (the keypoint representing the little finger's base) to the second metacarpal (the keypoint representing the index finger's base) lies along the yz-plane.

   This final alignment step ensures that the orientation of the hand is consistent across different samples.

Once the hands are normalized through these three steps, the keypoints are flattened into a 120-dimensional vector. This vector originates from the 20 3D keypoints for both hands (2 hands $\times$ 20 keypoints = 40 keypoints, each with 3 coordinates). In addition to the normalized keypoints, the original wrist positions and rotation matrices for both hands are concatenated to the feature vector, resulting in a final vector of 144 dimensions (120 from the keypoints, 6 from the wrist positions, and 18 from the rotation matrices).

This final 144-dimensional feature vector is used as input for the hand pose model described in 3.1.2.

Figure 4.7 illustrates the entire hand normalization procedure, including the translation, length modification, and rotation alignment steps.

## 4.3 Augmentation

We apply several data augmentation techniques to mitigate the risk of overfitting and enhance the generalization capabilities of our models. The following subsections provide a detailed description of these augmentations.

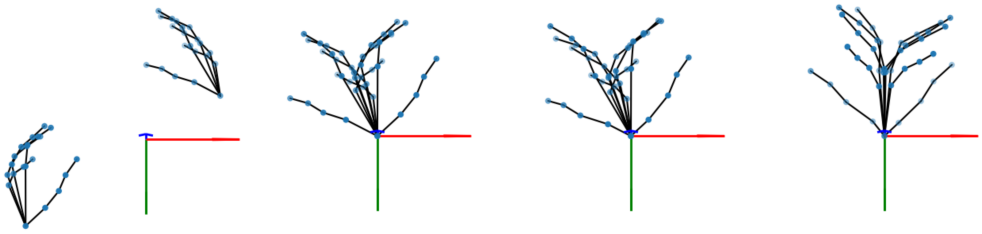

Figure 4.7: Example of hand pose normalization: From left to right, the original hand pose, the keypoints after translation to the origin, the keypoints after edge length modification, and the final result after rotation to a canonical orientation.

### 4.3.1 Mixup

First proposed in [44], Mixup generates a new data point $(\mathbf{x}, y)$ from data points $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$ by applying the following transformations:

$$\mathbf{x} = \lambda \mathbf{x}_i + (1 - \lambda)\mathbf{x}_j,$$

$$y = \lambda y_i + (1 - \lambda) y_j,$$

where $\lambda \sim Beta(\alpha, \alpha)$.

In this project, we utilize the timm library [60], which implements Mixup by symmetrically mixing the data. Given the original samples $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$, the new samples $(\tilde{\mathbf{x}}_i, \tilde{y}_i)$ and $(\tilde{\mathbf{x}}_j, \tilde{y}_j)$ are formed as follows:

$$\tilde{\mathbf{x}}_i = \lambda \mathbf{x}_i + (1 - \lambda)\mathbf{x}_j,$$

$$\tilde{y}_i = \lambda y_i + (1 - \lambda) y_j,$$

$$\tilde{\mathbf{x}}_j = \lambda \mathbf{x}_j + (1 - \lambda)\mathbf{x}_i,$$

$$\tilde{y}_j = \lambda y_j + (1 - \lambda) y_i.$$

This mixing of the data helps create more diverse training samples, which can improve the generalization ability of the model.

The second row of Figure 4.8 illustrates an example of Mixup.

### 4.3.2 CutMix

Let $\mathbf{x} \in \mathbb{R}^{W \times H \times C}$ and $y$ denote a training image and its label, respectively. CutMix [45] is an augmentation technique that generates a new sample $(\mathbf{x}, y)$ by combining two original samples $(\mathbf{x}_i, y_i)$ and $(\mathbf{x}_j, y_j)$ as follows:

$$\mathbf{x} = \mathbf{M} \odot \mathbf{x}_i + (\mathbf{1} - \mathbf{M}) \odot \mathbf{x}_j$$

$$y = \lambda y_i + (1 - \lambda) y_j,$$

where $\mathbf{M} \in \{0,1\}^{W \times H}$ is a binary mask, $\mathbf{1}$ is a binary mask filled with ones, $\odot$ denotes element-wise multiplication, and $\lambda \sim Beta(\alpha, \alpha)$.

In the timm library, CutMix is implemented by first sampling the bounding box coordinates $\mathbf{B} = (r_x, r_y, r_w, r_h)$ to define the cropping region for a batch of images. These coordinates are uniformly sampled as follows:

$$\begin{aligned} r_w &= W\sqrt{1-\lambda} \\ r_x &\sim \text{Unif } (r_w, W - r_w) \\ r_h &= H\sqrt{1-\lambda} \\ r_y &\sim \text{Unif } (r_h, H - r_h)\,. \end{aligned}$$

As a result, the ratio of the cropped area is equal to $1 - \lambda$. The binary mask $\mathbf{M} \in 0, 1^{W \times H}$ is generated by setting the values within the bounding box $\mathbf{B}$ to 0 and the values outside the box to 1. Finally, as with Mixup, the samples are combined in pairs.

The third row of Figure 4.8 shows an example of CutMix.

### 4.3.3 Random Erasing

Random Erasing [46] is an augmentation technique which consists of randomly selecting a rectangle region in the input image or video, and replace its pixels with random values or a fixed value. In this project, we utilize PyTorch [61] to apply random erasing, setting the pixels in the erased region to 0 across all channels. Figure 4.9 demonstrates an example of this transformation

### 4.3.4 Random Horizontal Flipping

Random horizontal flipping is a transformation that flips the image along the y-axis, as shown in Figure 4.10. We apply this transformation with a probability of 0.5 to introduce variability between right-handed and left-handed subjects, as the latter category is less represented. Additionally, the same transformation is applied to the hand pose data to ensure consistency between the modalities. Figure 4.11 displays horizontal flipping of the 3D hand keypoints.

### 4.3.5 RandAugment

First introduced in [43], RandAugment eliminates the need to search the space of possible augmentation strategies, which can complicate training and be computationally expensive. Instead, for each batch, RandAugment selects N transformations from a predefined set and applies them. The parameters of each transformation are derived from parameter-specific spaces based on the magnitude

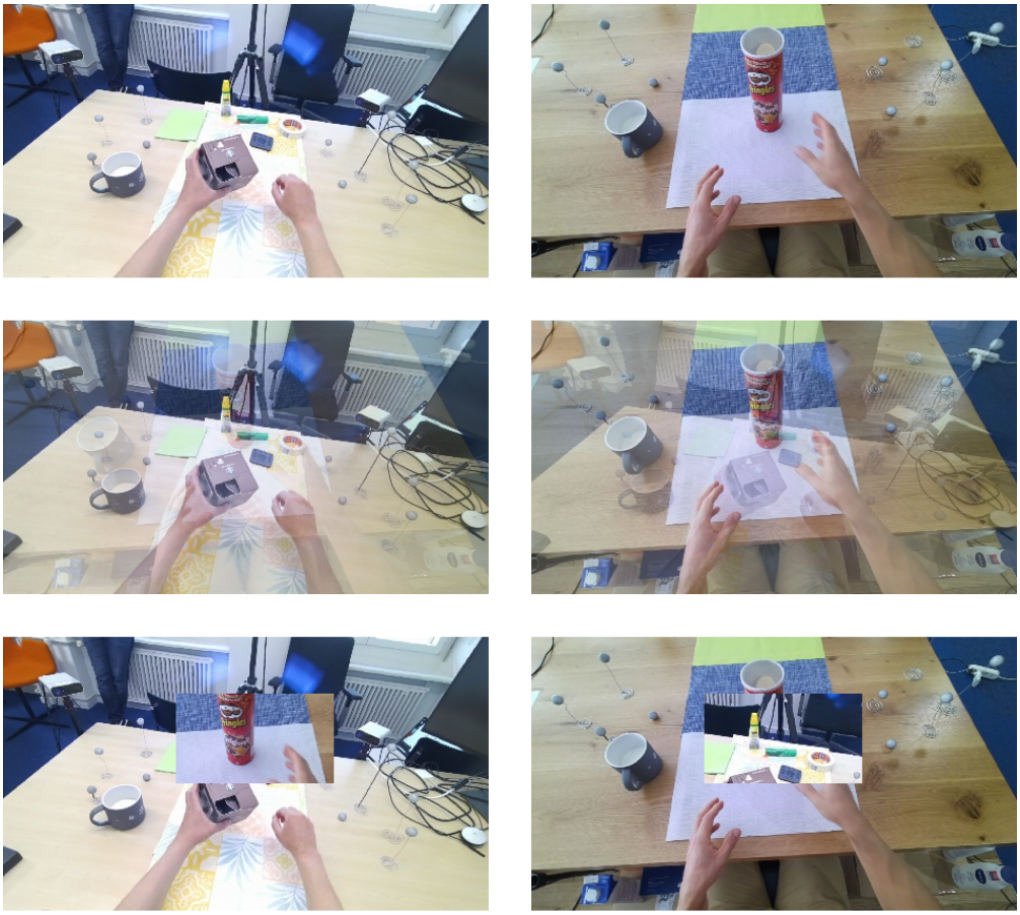

Figure 4.8: Mixup and CutMix examples. The first row displays the original images, the second row shows the results of Mixup, and the third row illustrates the outputs of CutMix.

Figure 4.9: Example of random erasing.

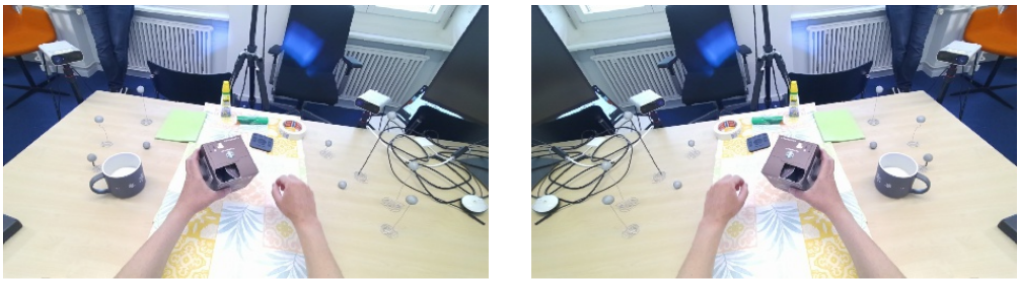

Figure 4.10: Example of horizontal flipping on images.

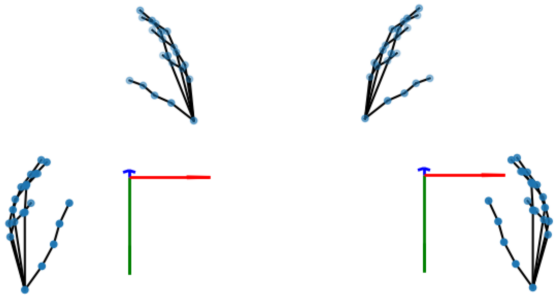

Figure 4.11: Examples of random horizontal flipping applied to the hand pose. From left to right: the original hand pose and the horizontally flipped hand pose.

M, where larger values of M correspond to more substantial transformations of the original sample. Figure 4.12 presents two examples of RandAugment applied with different values of M and N=2.

In our approach, RandAugment is also applied to the hand pose data, provided the selected transformation(s) are not specific to the RGB modality. When using both modalities together, the same augmentations are applied to both to ensure consistency. Figure 4.13 shows examples of transformations applied to the 3D hand keypoints.

### 4.3.6 Repeated Augmentation

In repeated augmentation [47], during each epoch, the model sees only a portion of the dataset, with samples being replicated multiple times. These repeated samples are then sent to different GPUs and augmented differently. This strategy improves model generalization and accelerates convergence by exposing the model to a wider variety of augmented views of the data.

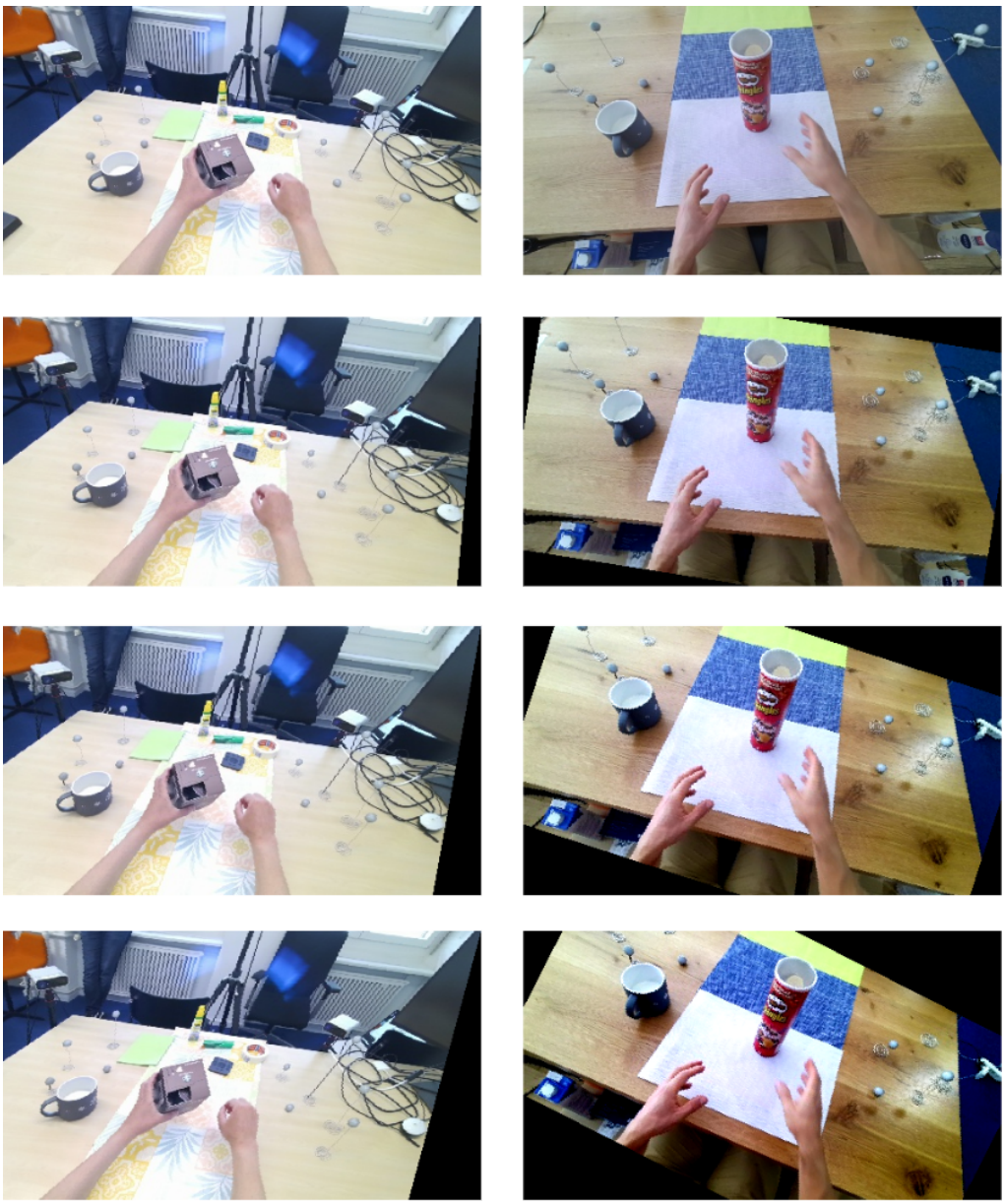

Figure 4.12: Examples of RandAugment. Each row displays the result of RandAugment with a different value of M: M=9 in row 2, M=18 in row 3, and M=27 in row 4. The transformations applied to the first image include shear along the x-axis and auto contrast, while the second image undergoes rotation and contrast adjustments.

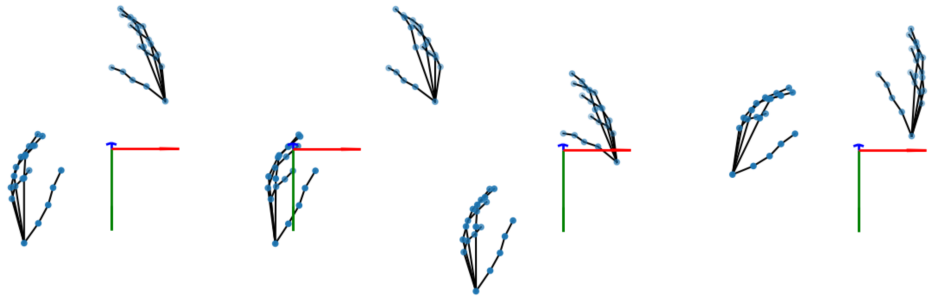

Figure 4.13: Examples of transformations applied independently to the hand pose. From left to right: the original hand pose, the hand pose translated along the x-axis, translated along the y-axis, and rotated.

## 4.4 Training Configuration and Evaluation Metrics

We train the models on the training set of the H2O dataset [33] and evaluate them on the validation set, as ground truth labels—which are not provided for the test set—are required for frame-level prediction. During training, we use the weighted cross-entropy loss, where the weight $W_c$ of class $c$ is computed as follows:

$$W_c = \frac{|\mathcal{X}|}{|\{(\mathbf{x}, y) \in \mathcal{X} \mid y = c\}|}.$$

The weights are subsequently normalized by dividing by the minimum weight, ensuring that all weights are 1-based.

As evaluation metrics, we use accuracy and the Macro F1-score. The Macro F1-score is calculated as the arithmetic mean of the per-class F1-scores, treating all classes equally regardless of their support. This makes it particularly suitable for assessing model performance on datasets with class imbalances, as it provides a balanced view across all classes. Additionally, we evaluate performance on segments, where segment predictions are obtained through majority voting over the single-frame predictions within the segment.

To complement the accuracy and F1-score evaluations, we also assess computational efficiency through inference speed and CPU usage. Inference speed is defined as the average time (in milliseconds) required for the model to process a single input (frame or sequence) on both GPU and CPU. For RGB sequence models, the input consists of a sequence of RGB frames within the model's context, sampled at the frequency used during training. CPU usage is evaluated as the time required to generate predictions for each time step in a sequence of 30 steps (one second), assuming a setting like the one depicted in Figure 4.4. In this scenario, at time step 8, for example, only the HP stream computes a new feature, as the RGB buffer remains unchanged. Additionally, CPU usage is reported as a percentage to facilitate easier comparison across models.

## 4.5 Experimental Conditions

Model training was performed on a SLURM cluster using either two or four GPUs, depending on availability. The GPUs used for training included the Titan RTX (24 GB memory), GeForce RTX 3090 (24 GB memory), and Tesla V100 (32 GB memory). For CPU inference speed and CPU usage, a single thread of an AMD EPYC 7742 CPU (base clock: 2.25 GHz) was utilized. GPU inference speed tests were conducted on a GeForce RTX 3090.

Chapter 5

# Results

In this chapter, we present a combined analysis of model performance, inference speed, and CPU usage. The results are organized based on model type to facilitate easier exploration. For a comprehensive overview of all models and experimental results, refer to Appendix B, where the complete tables can be found.

We evaluate the models on both frame-level and segment-level predictions, with the primary emphasis on frame-level action recognition. The names of sequence models follow the format $CxFyCwFz$, where $CxFy$ specifies the context length (in seconds) and frequency (in Hz) for the RGB modality, and $CwFz$ indicates the context length and frequency for the hand pose modality. A $v$ appended to $CxFyCwFz$ indicates that the model is trained to predict only the verb (i.e., the action type), rather than the full action. For instance, in this configuration, "grab book" and "grab lotion" are grouped into the same class.

We also analyze computational efficiency, including inference speed and CPU usage. For definitions of these metrics and details on how they are measured, refer to Section 4.4. The analysis is divided as follows: Section 5.1 describes the models that operate on a single frame, i.e., have a context length of 0 for both modalities; Section 5.2 presents the results for sequence models that process a single modality; Section 5.3 explores multimodal models; finally, Section 5.4 provides a comparison with some existing works.

## 5.1 Single-frame models

### Training Configuration

Both HP-MLP models were trained using the AdamW optimizer [62, 63] with a weight decay of 0.01 and an initial learning rate of $10^{-5}$, which is linearly increased to $10^{-4}$ over the first 10 epochs. We applied RandAugment and repeated augmentation as data augmentation techniques, and employed smoothing with a factor of 0.2 and a dropout rate of 0.5 for additional regularization. Both models

have a hidden size of 512.

RegNet-12GF was trained using the SGD optimizer with a momentum of 0.9 and a weight decay factor of 0.025. In contrast, the AdamW optimizer was used for both LeViT-256 models, with different hyperparameters: a weight decay factor of 0.025 and a fixed learning rate of 0.00005 for LeViT-256, and a weight decay factor of 0.005 with an initial learning rate of $10^{-4}$, which is linearly reduced to $10^{-5}$ over the first 10 epochs, for LeViT-256-distilled. Strong augmentation techniques were employed for all RGB models, including RandAugment, erasing with a probability of 0.4, repeated augmentation, as well as Mixup and CutMix with $\lambda \sim Beta(0.3, 0.3)$ and a probability of 0.8. To mitigate overfitting, label smoothing with a factor of 0.2 was applied, along with dropout before the classification head (with probability 0.5). Early stopping was also used when applying distillation. We employed hard distillation [50, 42] with $\alpha = 0.5$ and $\tau = 1$ to transfer the knowledge of RegNet-12GF to LeViT-256

Finally, the FusionNet model was trained using the AdamW optimizer with a weight decay factor of 0.01 and an initial learning rate of 0.00005, which is decayed to $10^{-5}$. Smoothing (0.2), dropout (0.5), and early stopping were used for regularization. All augmentations, except Mixup and CutMix, were also applied. For this analysis, LeViT-256 and HP-MLP-v serve as feature extractors for the RGB and hand pose modalities, respectively, and both were frozen during training.

### Performance and Efficiency Analysis

Table 5.1 presents the performance and inference time for models that operate on a single frame. We can observe that the RegNet-12GF has higher F1-score and accuracy than LeViT-256, but at the cost of more than seven times longer inference time on CPU. This makes it unsuitable for deployment in resource-constrained environments, such as wearable devices like AR glasses.

Distillation helps bridge this performance gap, as demonstrated by the results of LeViT-256-distilled. Knowledge distillation improves the F1-score from 70.584% to 73.692%, while maintaining the same inference speed as the original LeViT-256. Consequently, we select the LeViT-256-distilled model as the feature extractor for the RGB modality.

We also experiment with the HP-MLP model for both verb and action prediction tasks. The results indicate that while the model performs well on verb prediction, it achieves significantly lower evaluation metrics for action prediction. This can be attributed to the hand pose modality lacking information about the object being manipulated. The first two rows of Table 5.1b present the inference time for HP-MLP. These models demonstrate extremely fast inference, making them well-suited for practical deployment.

Finally, we analyze the effect of single-frame fusion using the FusionNet model described in Section 3.2. As presented in Table 5.1a, FusionNet achieves a significant performance improvement over LeViT-256-distilled, increasing the F1-score from 73.692% to 76.812%. This result highlights the complementary nature of the RGB and hand pose modalities in the single-frame setting.

| Model | Frame Accuracy (%) | Frame F1-score (%) | Segment Accuracy (%) | Segment F1-score (%) |
|---|---|---|---|---|
| HP-MLP-v | 78.373 | 79.686 | 80.328 | 85.748 |
| HP-MLP | 52.079 | 46.933 | 53.279 | 51.098 |
| RegNet | 79.997 | 74.844 | 81.967 | 79.437 |
| LeViT-256 | 76.499 | 70.584 | 78.689 | 76.993 |
| LeViT-256-distilled | 79.112 | 73.692 | 81.148 | 79.854 |
| FusionNet | 81.139 | 76.812 | 84.426 | 83.266 |

(a) Accuracy and F1-score for frame-level and segment-level predictions.

| Model | Time GPU mean (ms) | Time GPU std (ms) | Time CPU mean (ms) | Time CPU std (ms) |
|---|---|---|---|---|
| HP-MLP-v | 0.332 | 0.016 | 0.214 | 0.010 |
| HP-MLP | 0.330 | 0.007 | 0.215 | 0.003 |
| RegNet-12GF | 8.961 | 0.052 | 336.472 | 2.098 |
| LeViT-256 | 8.615 | 0.189 | 45.988 | 1.244 |
| LeViT-256-distilled | 8.571 | 0.157 | 44.986 | 0.691 |
| FusionNet | 9.223 | 0.231 | 46.331 | 0.564 |

(b) Inference time.

Table 5.1: Performance and inference time for single-frame models.

## 5.2 Unimodal Sequence Models

### 5.2.1 RGB Models

#### Training Configuration

All sequence models that operate on RGB frames were trained using nearly identical training configurations. We utilized the AdamW optimizer with a weight decay of 0.005 and an initial learning rate of $10^{-4}$, which is decayed linearly to $10^{-5}$ over the first 10 epochs. The same augmentation and regularization techniques described in Section 5.1 were employed for these models. The batch size was varied slightly across models depending on the input sequence frequency.

All TMLPs consist of 12 layers with decreasing hidden sizes: the first 8 layers have a hidden size of 512, the next one has 384, and the final two have 256. The TCNs consist of 6 blocks: the first 4 blocks have a channel size of 512, the next one has 384, and the last one has 256.

### Performance and Efficiency Analysis

The performance, inference time, and CPU usage for sequence models processing RGB frames are summarized in Table 5.2. We observe a steady decline in F1-score from TMLP-C2F30C0F0 to TMLP-C2F1C0F0, which corresponds to a reduction in the input sequence's sampling frequency. However, the reduction in CPU inference time and usage is considerably more pronounced. Notably, both CPU inference time and usage decrease by approximately a factor of three each time the sampling frequency is reduced by a factor of three. The key takeaway is that we can reduce the RGB frequency to match the resource constraints of the environment without a substantial loss in performance.

We also present the performance of two TCN models at frequencies of 3 Hz and 1 Hz. Since TCNs have a significantly larger number of hyperparameters compared to TMLPs, we choose to continue the analysis with the latter.

| Model | Frame Accuracy (%) | Frame F1-score (%) | Segment Accuracy (%) | Segment F1-score (%) |
|---|---|---|---|---|
| TMLP-C2F30C0F0 | 92.327 | 91.154 | 95.082 | 95.352 |
| TMLP-C2F10C0F0 | 90.454 | 88.974 | 90.984 | 91.084 |
| TMLP-C2F3C0F0 | 88.589 | 86.667 | 88.525 | 88.674 |
| TMLP-C2F1C0F0 | 85.900 | 82.809 | 87.705 | 87.322 |
| TCN-C2F3C0F0 | 86.621 | 82.752 | 87.705 | 86.781 |
| TCN-C2F1C0F0 | 85.556 | 82.232 | 89.344 | 89.332 |

(a) Accuracy and F1-score for frame-level and segment-level predictions.

| Model | Time GPU mean (ms) | Time GPU std (ms) | Time CPU mean (ms) | Time CPU std (ms) |
|---|---|---|---|---|
| TMLP-C2F30C0F0 | 22.028 | 0.203 | 2320.444 | 6.069 |
| TMLP-C2F10C0F0 | 16.200 | 0.279 | 720.304 | 2.470 |
| TMLP-C2F3C0F0 | 16.072 | 0.272 | 219.823 | 1.272 |
| TMLP-C2F1C0F0 | 16.180 | 0.268 | 82.654 | 0.653 |
| TCN-C2F3C0F0 | 13.682 | 0.241 | 251.176 | 1.950 |
| TCN-C2F1C0F0 | 13.652 | 0.192 | 100.226 | 0.742 |

(b) Inference time.

| Model | Time CPU mean (ms) | Time CPU std (ms) | Usage (%) |
|---|---|---|---|
| TMLP-C2F30C0F0 | 1832.083 | 12.922 | 183.208 |
| TMLP-C2F10C0F0 | 537.309 | 3.364 | 53.731 |
| TMLP-C2F3C0F0 | 154.724 | 1.670 | 15.472 |
| TMLP-C2F1C0F0 | 50.596 | 0.510 | 5.060 |
| TCN-C2F3C0F0 | 906.011 | 2.716 | 90.691 |
| TCN-C2F1C0F0 | 725.977 | 1.348 | 72.598 |

(c) CPU usage.

Table 5.2: Performance, inference time, and CPU usage for RGB sequence models.

### 5.2.2 Hand Pose Models

#### Training Configuration

The sequence models that process the hand pose were trained using identical configurations. We used the AdamW optimizer with a weight decay factor of 0.005 and an initial learning rate of 0.00005, which is reduced linearly to $10^{-5}$ over the first 10 epochs. RandAugment and repeated augmentation, alongside label smoothing with a factor of 0.2, early stopping with a patience of 20 epochs, and dropout with a probability of 0.5 before the classification head, were applied to reduce overfitting. The batch size was kept constant across models. Unlike the RGB sequence models, the HP-MLP-v feature extractor was kept frozen during training in all experiments.

All TMLPs have 6 layers with a fixed hidden size of 512 dimensions.

#### Performance and Efficiency Analysis

The performance, inference time, and CPU usage for sequence models processing hand pose samples are presented in Table 5.3. Interestingly, unlike the RGB sequence models, we do not observe the same steady decline in performance as the frequency is reduced. In fact, the performance metrics slightly improve from TMLP-C0F0C2F30-v to TMLP-C0F0C2F10-v when evaluated on the frame-level prediction task. We hypothesize that the shallow TMLP struggles to capture meaningful motion when presented with a high-frequency sequence, where consecutive hand poses show minimal change. Reducing the frequency allows the model to focus on more significant temporal differences, leading to slight improved performance.

Table 5.3c shows a much larger drop in CPU usage from 30 Hz to 10 Hz compared to Table 5.2c. This difference arises because, for the RGB modality, the feature extractor is the bottleneck, while the inference time of the TMLP on top is comparatively low. As a result, reducing the sampling frequency by a factor of three leads to an almost equal reduction in CPU usage. In contrast, for the hand pose modality, the bottleneck is the TMLP. The TMLP that operates over 20 time steps is already twice as fast on CPU as the TMLP that processes the full 60 samples, as shown in Table 5.3b. Additionally, the TMLP is executed only when a new data point is sampled, resulting in a reduction in CPU usage by a factor of approximately six from TMLP-C0F0C2F30-v to TMLP-C0F0C2F10-v. In practice, we can choose to sample every hand pose while still using 20 out of 60 data points, evenly distributed across the context, to make the prediction. This would reduce CPU usage by a factor of three while eliminating the need to replicate predictions during periods where no new hand pose data point is sampled.

| Model | Frame Accuracy (%) | Frame F1-score (%) | Segment Accuracy (%) | Segment F1-score (%) |
|---|---|---|---|---|
| TMLP-C0F0C2F30-v | 86.157 | 85.301 | 87.705 | 88.761 |
| TMLP-C0F0C2F10-v | 86.810 | 86.348 | 87.705 | 88.889 |
| TMLP-C0F0C2F3-v | 85.118 | 84.848 | 84.426 | 85.356 |
| TMLP-C0F0C2F1-v | 84.087 | 82.850 | 83.607 | 85.103 |

(a) Accuracy and F1-score for frame-level and segment-level predictions.

| Model | Time GPU mean (ms) | Time GPU std (ms) | Time CPU mean (ms) | Time CPU std (ms) |
|---|---|---|---|---|
| TMLP-C0F0C2F30-v | 4.118 | 0.017 | 10.455 | 0.309 |
| TMLP-C0F0C2F10-v | 4.089 | 0.027 | 5.428 | 0.064 |
| TMLP-C0F0C2F3-v | 4.002 | 0.021 | 3.938 | 0.294 |
| TMLP-C0F0C2F1-v | 4.024 | 0.027 | 3.136 | 0.036 |

(b) Inference time.

| Model | Time CPU mean (ms) | Time CPU std (ms) | Usage (%) |
|---|---|---|---|
| TMLP-C0F0C2F30-v | 282.789 | 1.432 | 28.279 |
| TMLP-C0F0C2F10-v | 52.036 | 0.148 | 5.204 |
| TMLP-C0F0C2F3-v | 11.330 | 0.072 | 1.133 |
| TMLP-C0F0C2F1-v | 3.207 | 0.036 | 0.321 |

(c) CPU usage.

Table 5.3: Performance, inference time, and CPU usage for Hand Pose sequence models.

## 5.3 Multimodal Sequence Models

### Training Configuration

The MM-TMLP models were all trained using the AdamW optimizer with a weight decay factor of 0.01 and an initial learning rate of $10^{-5}$, which is reduced linearly to $10^{-6}$ within the first 5 epochs. All augmentations and regularization techniques described in Section 5.1 were utilized, except for Mixup and CutMix. We also employed early stopping with a patience of 10 epochs. The batch size is set to 256 across models.

The MM-TMLPs are created following the framework described in Section 3.3.3, where the modality-specific TMLPs are those detailed in Sections 5.2.1 and 5.2.2. For the hand pose stream, we use TMLPs trained to predict verbs, hoping to obtain more motion context from their feature representations. For the fusion module, we use a hidden size of 512, BatchNorm, and a Leaky ReLU activation function. In all experiments, only this fusion head is trained.

### Performance and Efficiency Analysis

Table 5.4a presents the results for the MM-TMLP models across all combinations of RGB and hand pose sampling frequencies. The corresponding F1-scores for the MM-TMLP models are also shown in Figure 5.1, alongside those of the RGB TMLP models. A comparison with Table 5.2a shows that, for a fixed $f_{RGB}$, incorporating the hand pose stream consistently enhances performance, irrespective of the hand pose sampling frequency $f_{HP}$. Additionally, for a fixed $f_{RGB}$, the macro F1-score remains relatively stable as $f_{HP}$ decreases, though a more significant decline is observed when $f_{HP} = 1$.

For a fixed $f_{HP}$, the drop in the F1-score becomes more pronounced as $f_{RGB}$ decreases, with the largest difference observed being 7.113 for $f_{HP} = 1$. This behavior is expected, as in the unimodal setting, reducing $f_{RGB}$ from 30 to 1 results in a significantly larger drop than the relatively smaller decline observed when reducing $f_{HP}$ from 30 to 1 (see Tables 5.2a and 5.3a). Despite this, the hand pose modality helps to narrow the performance gap between the model with $f_{RGB} = 30$ and the model with $f_{RGB} = 1$. When using only RGB frames, the F1-score difference is 8.345, whereas incorporating both modalities reduces the maximum difference to 7.113 (observed at $f_{HP} = 3$), with a mean difference of 6.164. Additionally, the performance gap between the best-performing model (MM-TMLP-C2F30C2F3) and the worst-performing model (MM-TMLP-C2F1C2F1) is 7.867, which is slightly smaller than the 8.345 observed with RGB alone.

Moving to inference speed and CPU usage, which are reported in Tables 5.4b and 5.4c, and displayed in Figures 5.2 and 5.3, we observe that the RGB modality has a significant impact on both measurements. Specifically, we note a trend of reduction in both inference speed and CPU usage by approximately a factor of three for every reduction in $f_{RGB}$, a trend that we also observed for the RGB-only models.

Focusing on CPU usage, a comparison between Table 5.2c and Table 5.4c reveals that, for a fixed $f_{RGB}$, adding the hand pose stream to the RGB stream always increases the usage, with the increase being larger for higher values of $f_{HP}$. This is the expected behavior.

The results are promising, demonstrating that by reducing the sampling rate of RGB frames from 30 Hz to 10 Hz, we can consistently generate predictions at 30 Hz on a single thread of an AMD EPYC 7742 CPU without exceeding the 100% usage threshold. If we consider, for instance, models MM-TMLP-C2F30C2F30 and MM-TMLP-C2F10C2F30, a 61.11% reduction in CPU usage corresponds to a 0.835% reduction in F1-score.

To conclude the analysis on prediction performance, Table 5.5 presents the results of the MM-TMLP models when sequences are generated as depicted in Figure 4.4. These results are also illustrated in Figure 5.4. This setup simulates

| Model | Frame Accuracy (%) | Frame F1-score (%) | Segment Accuracy (%) | Segment F1-score (%) |
|---|---|---|---|---|
| MM-TMLP-C2F30C2F30 | 93.573 | 92.449 | 95.082 | 95.202 |
| MM-TMLP-C2F10C2F30 | 92.421 | 91.677 | 95.082 | 95.228 |
| MM-TMLP-C2F3C2F30 | 89.844 | 88.055 | 90.164 | 90.318 |
| MM-TMLP-C2F1C2F30 | 89.414 | 87.209 | 92.623 | 92.349 |
| MM-TMLP-C2F30C2F10 | 93.487 | 92.527 | 96.721 | 97.107 |
| MM-TMLP-C2F10C2F10 | 92.696 | 91.595 | 94.262 | 94.285 |
| MM-TMLP-C2F3C2F10 | 90.419 | 88.577 | 92.623 | 92.193 |
| MM-TMLP-C2F1C2F10 | 89.457 | 86.540 | 92.623 | 92.349 |
| MM-TMLP-C2F30C2F3 | 93.813 | 92.887 | 95.902 | 96.154 |
| MM-TMLP-C2F10C2F3 | 92.404 | 91.123 | 92.623 | 92.943 |
| MM-TMLP-C2F3C2F3 | 90.084 | 88.499 | 93.443 | 93.324 |
| MM-TMLP-C2F1C2F3 | 88.941 | 86.572 | 91.803 | 91.703 |
| MM-TMLP-C2F30C2F1 | 93.246 | 92.133 | 95.082 | 95.535 |
| MM-TMLP-C2F10C2F1 | 92.026 | 90.520 | 91.803 | 91.991 |
| MM-TMLP-C2F3C2F1 | 88.555 | 86.756 | 88.525 | 88.346 |
| MM-TMLP-C2F1C2F1 | 87.661 | 85.020 | 90.984 | 90.890 |

(a) Accuracy and F1-score for frame-level and segment-level predictions.

| Model | Time GPU mean (ms) | Time GPU std (ms) | Time CPU mean (ms) | Time CPU std (ms) |
|---|---|---|---|---|
| MM-TMLP-C2F30C2F30 | 25.891 | 0.246 | 2306.591 | 3.887 |
| MM-TMLP-C2F10C2F30 | 20.330 | 0.274 | 739.656 | 2.359 |
| MM-TMLP-C2F3C2F30 | 20.245 | 0.285 | 229.074 | 1.265 |
| MM-TMLP-C2F1C2F30 | 20.377 | 0.271 | 93.523 | 0.562 |
| MM-TMLP-C2F30C2F10 | 25.832 | 0.305 | 2292.625 | 3.846 |
| MM-TMLP-C2F10C2F10 | 20.305 | 0.271 | 722.962 | 2.351 |
| MM-TMLP-C2F3C2F10 | 20.222 | 0.285 | 224.186 | 1.283 |
| MM-TMLP-C2F1C2F10 | 20.323 | 0.267 | 88.688 | 0.684 |
| MM-TMLP-C2F30C2F3 | 25.799 | 0.228 | 2291.908 | 3.996 |
| MM-TMLP-C2F10C2F3 | 20.231 | 0.273 | 733.177 | 2.277 |
| MM-TMLP-C2F3C2F3 | 20.125 | 0.267 | 223.906 | 1.266 |
| MM-TMLP-C2F1C2F3 | 20.245 | 0.267 | 87.103 | 0.742 |
| MM-TMLP-C2F30C2F1 | 25.780 | 0.240 | 2301.329 | 4.333 |
| MM-TMLP-C2F10C2F1 | 20.277 | 0.275 | 724.087 | 2.416 |
| MM-TMLP-C2F3C2F1 | 20.162 | 0.268 | 222.389 | 1.256 |
| MM-TMLP-C2F1C2F1 | 20.318 | 0.268 | 86.358 | 0.743 |

(b) Inference time.

| Model | Time CPU mean (ms) | Time CPU std (ms) | Usage (%) |
|---|---|---|---|
| MM-TMLP-C2F30C2F30 | 2118.842 | 1.193 | 211.884 |
| MM-TMLP-C2F10C2F30 | 823.950 | 0.859 | 82.395 |
| MM-TMLP-C2F3C2F30 | 444.983 | 0.549 | 44.498 |
| MM-TMLP-C2F1C2F30 | 339.411 | 0.748 | 33.941 |
| MM-TMLP-C2F30C2F10 | 1880.565 | 1.001 | 188.056 |
| MM-TMLP-C2F10C2F10 | 588.946 | 0.769 | 58.895 |
| MM-TMLP-C2F3C2F10 | 207.478 | 0.337 | 20.748 |
| MM-TMLP-C2F1C2F10 | 103.454 | 0.202 | 10.345 |
| MM-TMLP-C2F30C2F3 | 1842.905 | 0.900 | 184.290 |
| MM-TMLP-C2F10C2F3 | 548.006 | 0.617 | 54.801 |
| MM-TMLP-C2F3C2F3 | 165.934 | 0.317 | 16.593 |
| MM-TMLP-C2F1C2F3 | 62.461 | 0.146 | 6.246 |
| MM-TMLP-C2F30C2F1 | 1833.635 | 0.957 | 183.363 |
| MM-TMLP-C2F10C2F1 | 540.224 | 0.624 | 54.022 |
| MM-TMLP-C2F3C2F1 | 157.785 | 0.282 | 15.778 |
| MM-TMLP-C2F1C2F1 | 53.618 | 0.176 | 5.362 |

(c) CPU usage.

Table 5.4: Performance, inference time, and CPU usage for MM-TMLP models.

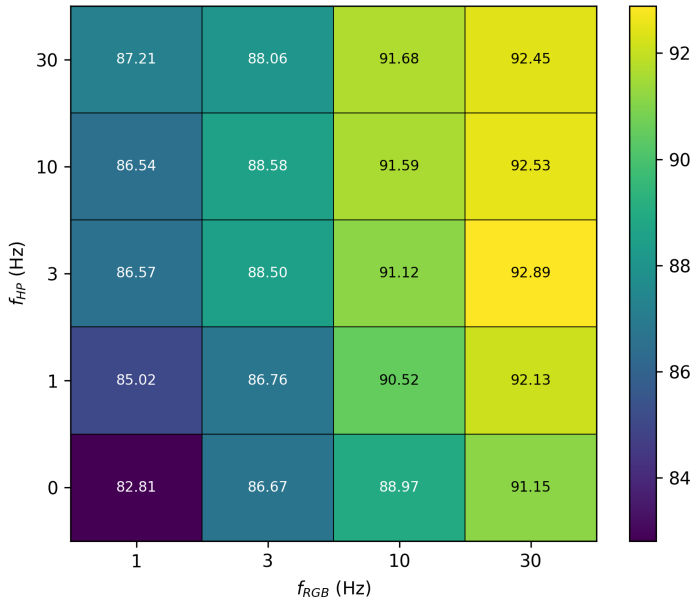


Figure 5.1: F1-scores of RGB TMLPs and MM-TMLPs for different combinations of RGB and hand pose sampling frequencies.

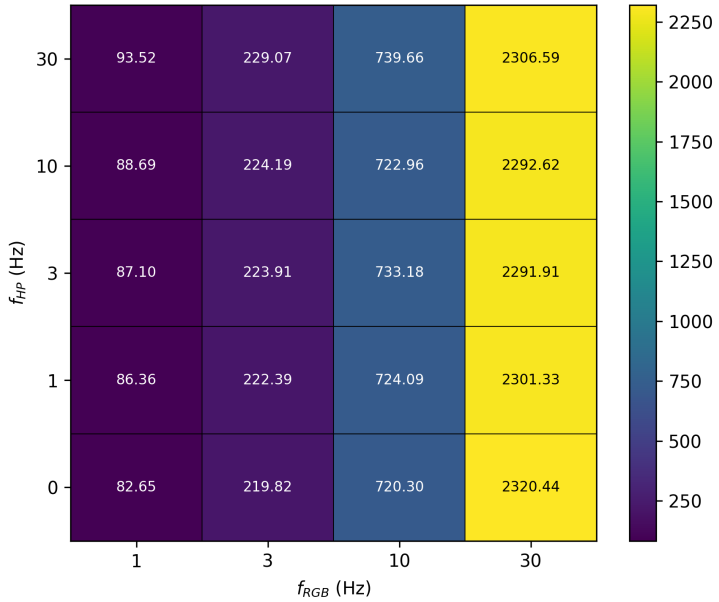


Figure 5.2: CPU inference time (in ms) of RGB TMLPs and MM-TMLPs for different combinations of RGB and hand pose sampling frequencies.

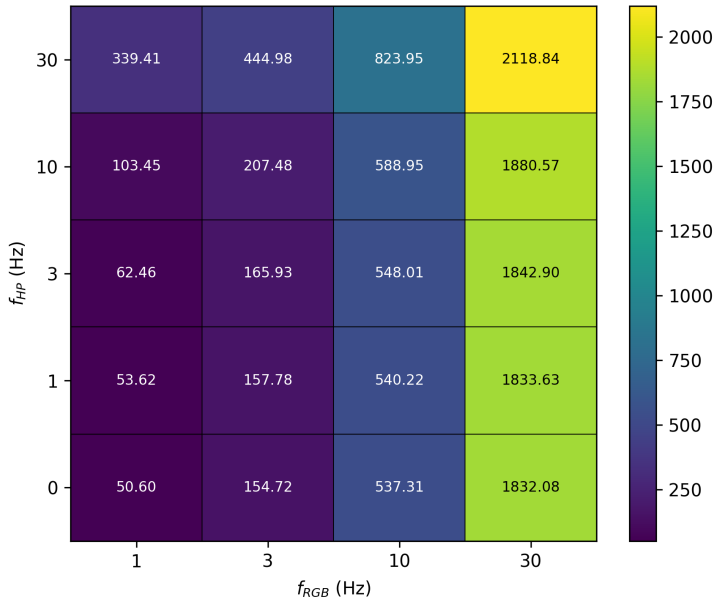


Figure 5.3: CPU usage (in ms) of RGB TMLPs and MM-TMLPs for different combinations of RGB and hand pose sampling frequencies.

an online scenario, where buffers maintain the most recently accepted samples from both modalities.

We observe that the performance either remains stable or declines only slightly. However, models with lower sampling frequencies tend to suffer the most from misalignments. For instance, the F1-score drops from 86.756 to 85.469 for MM-TMLP-C2F3C2F1, from 86.572 to 85.567 for MM-TMLP-C2F1C2F3, and from 85.02 to 82.138 for MM-TMLP-C2F1C2F1. Such low sampling frequencies for both modalities could significantly hinder performance in real-world scenarios, especially if actions change rapidly.

Models like MM-TMLP-C2F10C2F30 and MM-TMLP-C2F10C2F10 provide a good balance of performance, CPU usage, and robustness to misalignments, making them the most suitable choices in most situations. Under stricter resource constraints, other models can be selected, albeit at the cost of reduced performance.

| Model | Frame Accuracy (%) | Frame F1-score (%) | Segment Accuracy (%) | Segment F1-score (%) |
|---|---|---|---|---|
| MM-TMLP-C2F30C2F30 | 93.573 | 92.449 | 95.082 | 95.202 |
| MM-TMLP-C2F10C2F30 | 92.567 | 91.867 | 95.082 | 95.228 |
| MM-TMLP-C2F3C2F30 | 89.560 | 87.787 | 89.344 | 89.365 |
| MM-TMLP-C2F1C2F30 | 88.022 | 86.329 | 91.803 | 91.298 |
| MM-TMLP-C2F30C2F10 | 93.444 | 92.476 | 96.721 | 97.107 |
| MM-TMLP-C2F10C2F10 | 92.791 | 91.806 | 93.443 | 93.670 |
| MM-TMLP-C2F3C2F10 | 90.591 | 88.899 | 93.443 | 93.451 |
| MM-TMLP-C2F1C2F10 | 88.031 | 85.939 | 90.984 | 90.523 |
| MM-TMLP-C2F30C2F3 | 93.435 | 92.405 | 95.082 | 95.202 |
| MM-TMLP-C2F10C2F3 | 92.181 | 90.950 | 91.803 | 92.044 |
| MM-TMLP-C2F3C2F3 | 89.388 | 88.113 | 91.803 | 92.196 |
| MM-TMLP-C2F1C2F3 | 87.421 | 85.567 | 91.803 | 91.657 |
| MM-TMLP-C2F30C2F1 | 92.997 | 91.944 | 94.262 | 94.843 |
| MM-TMLP-C2F10C2F1 | 91.098 | 89.748 | 91.803 | 91.991 |
| MM-TMLP-C2F3C2F1 | 86.931 | 85.469 | 87.705 | 87.647 |
| MM-TMLP-C2F1C2F1 | 84.267 | 82.138 | 90.984 | 91.203 |

Table 5.5: Performance, inference time, and CPU usage for MM-TMLP models, as evaluated in the scenario depicted in Figure 4.4.

Finally, Figure 5.5 illustrates how the F1-score of models trained on the action prediction task varies with inference speed. The x-axis is on a logarithmic scale. The dotted line represents a linear increase in F1-score, ranging from 0 at no inference time to the maximum observed F1-score and inference time. Sequence models are color-coded based on their $f_{RGB}$ value to emphasize the clusters. We observe that the models lie above the line, confirming that reducing inference time does not lead to a proportional decrease in performance. Even for the HP-MLP model with near-zero inference time, we still achieve results better than random.

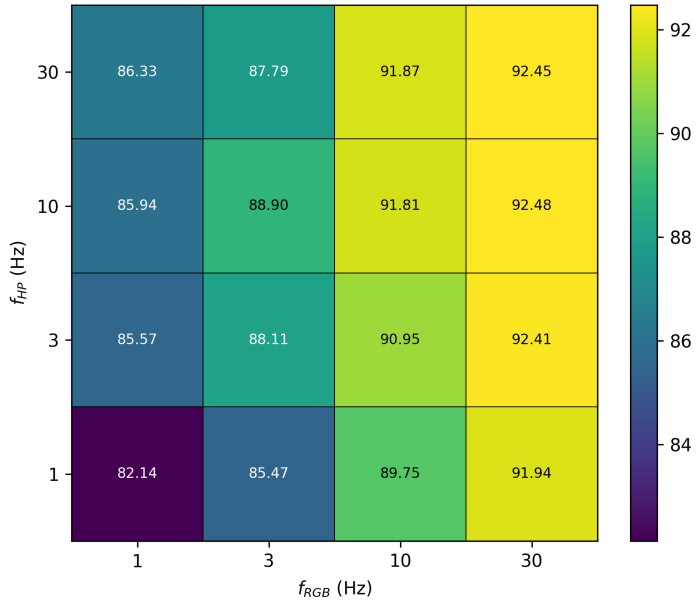


Figure 5.4: F1-scores of MM-TMLPs for different combinations of RGB and hand pose sampling frequencies, as evaluated in the scenario depicted in Figure 4.4.

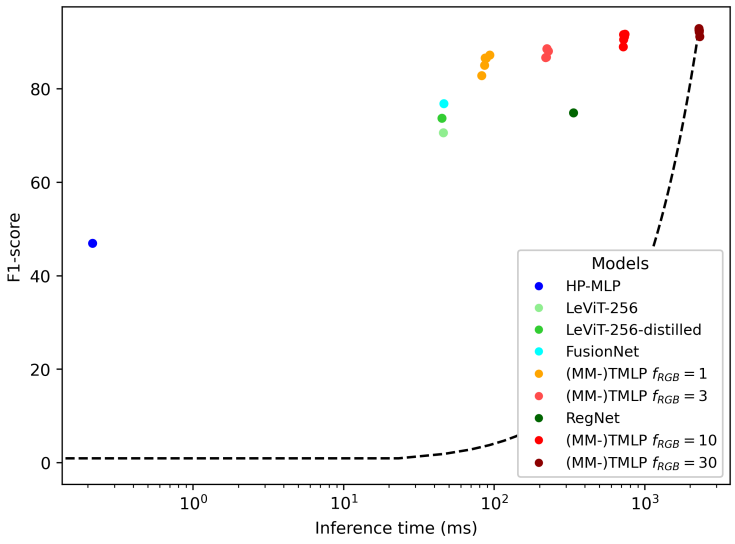


Figure 5.5: F1-score vs. Inference time for the action prediction models. The x-axis is in logarithmic scale, and the dotted line indicates a linear increase in F1-score.

## 5.4 Existing Works

We compare two of our MM-MLPs with existing methods that report results on the validation set of the H2O dataset. Table 5.6 presents the segment-level accuracy for these models. Despite being trained to predict actions for individual frames—a task requiring solutions distinct from predicting actions for entire segments—our models demonstrate competitive performance on the validation set.

| **Model** | **Segment Accuracy** |
|---|---|
| C2D [64] | 76.10 |
| I3D [65] | 85.15 |
| SlowFast [66] | 86.00 |
| H+O [67] | 80.49 |
| ST-GCN [11] | 83.47 |
| TA-GCN [33] | 86.78 |
| H2OTR [68] | 92.60 |
| MM-TMLP-C2F30C2F10 (best) | 96.72 |
| MM-TMLP-C2F10C2F30 | 95.08 |

Table 5.6: Segment-level accuracy on the validation set of the H2O dataset.

Chapter 6

# Conclusion and Future Work

In this work, we presented an architectural framework to address frame-level egocentric action prediction in resource-constrained environments such as augmented reality glasses. Our findings demonstrated that models incorporating contextual information achieve higher prediction performance at the current time step compared to those processing data sampled solely at that step.

Additionally, we showed that leveraging both hand pose and RGB modalities enhances the performance of models that rely only on RGB, with minimal impact on inference speed and CPU usage. Further analysis focused on combinations of sampling frequencies for the two modalities, revealing that performance drops—measured as reductions in macro F1-score on—are significantly smaller than the corresponding reductions in inference speed and, critically, CPU usage.

These results justify the selection of models like MM-TMLP-C2F10C2F30 over higher-performance alternatives such as MM-TMLP-C2F30C2F30. By sacrificing a small amount of predictive performance, MM-TMLP-C2F10C2F30 achieves CPU usage below the 100% threshold on a single thread of an AMD EPYC 7742 CPU, making it more practical for deployment in real-world scenarios. Additionally, we demonstrated that MM-TMLP-C2F10C2F30—and other models with an RGB frequency of 10 Hz or lower—maintain robust performance under more realistic, real-world conditions, confirming their practicality and adaptability.

Future work should focus on improving the efficiency and adaptability of the proposed framework. One direction is optimizing the RGB stream, potentially through lightweight architectures or quantization techniques, to reduce computational costs further. Another key area is the creation of a large, specialized dataset for egocentric single-frame action recognition, addressing the current lack of data for this specific use case.

Additionally, improving robustness to modality misalignments could enhance real-world performance. Exploring new modalities, such as audio, gaze tracking, and head pose could unlock further performance gains. Finally, real-world deployment and testing of these models on AR glasses would provide valuable insights, ensuring practical applicability in diverse, challenging environments.

# Bibliography


[1] A. Núñez Marcos, G. Azkune, and I. Arganda-Carreras, "Egocentric vision-based action recognition: A survey," *Neurocomput.*, vol. 472, no. C, p. 175–197, Feb. 2022. [Online]. Available: https://doi.org/10.1016/j.neucom.2021.11.081

[2] H. Pirsiavash and D. Ramanan, "Detecting activities of daily living in first-person camera views," in *2012 IEEE conference on computer vision and pattern recognition*. IEEE, 2012, pp. 2847–2854.

[3] K. Matsuo, K. Yamada, S. Ueno, and S. Naito, "An attention-based activity recognition for egocentric video," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2014.

[4] G. Kapidis, R. Poppe, E. van Dam, L. P. Noldus, and R. C. Veltkamp, "Object detection-based location and activity classification from egocentric videos: A systematic analysis," *Smart Assisted Living: Toward An Open Smart-Home Infrastructure*, pp. 119–145, 2020.

[5] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2017, pp. 7263–7271.

[6] S. Sudhakaran and O. Lanz, "Convolutional long short-term memory networks for recognizing first person interactions," in *Proceedings of the IEEE International Conference on Computer Vision (ICCV) Workshops*, Oct 2017.

[7] X. Shi, Z. Chen, H. Wang, D.-Y. Yeung, W.-K. Wong, and W.-c. Woo, "Convolutional lstm network: A machine learning approach for precipitation nowcasting," *Advances in neural information processing systems*, vol. 28, 2015.

[8] Y. Poleg, A. Ephrat, S. Peleg, and C. Arora, "Compact cnn for indexing egocentric videos," in *2016 IEEE winter conference on applications of computer vision (WACV)*. IEEE, 2016, pp. 1–9.

[9] S. Bambach, S. Lee, D. J. Crandall, and C. Yu, "Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions," in *Proceedings of the IEEE international conference on computer vision*, 2015, pp. 1949–1957.

[10] P. Das and A. Ortega, "Symmetric sub-graph spatio-temporal graph convolution and its application in complex activity recognition," in *ICASSP 2021-2021 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. IEEE, 2021, pp. 3215–3219.

[11] S. Yan, Y. Xiong, and D. Lin, "Spatial temporal graph convolutional networks for skeleton-based action recognition," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 32, no. 1, 2018.

[12] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," *Advances in neural information processing systems*, vol. 27, 2014.

[13] H. Kwon, Y. Kim, J. S. Lee, and M. Cho, "First person action recognition via two-stream convnet with long-term fusion pooling," *Pattern Recognition Letters*, vol. 112, pp. 161–167, 2018.

[14] Y. Li, M. Liu, and J. M. Rehg, "In the eye of beholder: Joint learning of gaze and actions in first person video," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[15] M. Lu, Z.-N. Li, Y. Wang, and G. Pan, "Deep attention network for egocentric action recognition," *IEEE Transactions on Image Processing*, vol. 28, no. 8, pp. 3703–3713, 2019.

[16] M. Lu, D. Liao, and Z.-N. Li, "Learning spatiotemporal attention for egocentric action recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV) Workshops*, Oct 2019.

[17] S. Sudhakaran, S. Escalera, and O. Lanz, "Lsta: Long short-term attention for egocentric action recognition," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 9954–9963.

[18] S. Sudhakaran and O. Lanz, "Attention is all we need: Nailing down object-centric attention for egocentric activity recognition," *arXiv preprint arXiv:1807.11794*, 2018.

[19] E. H. Spriggs, F. De La Torre, and M. Hebert, "Temporal segmentation and activity classification from first-person sensing," in *2009 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*. IEEE, 2009, pp. 17–24.

[20] Y. Lu and S. Velipasalar, "Human activity classification incorporating egocentric video and inertial measurement unit data," in *2018 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. IEEE, 2018, pp. 429–433.

[21] M. S. Shamil, D. Chatterjee, F. Sener, S. Ma, and A. Yao, "On the utility of 3d hand poses for action recognition," in *European Conference on Computer Vision*. Springer, 2025, pp. 436–454.

[22] A. Furnari and G. M. Farinella, "What would you expect? anticipating egocentric actions with rolling-unrolling lstms and modality attention," in *Proceedings of the IEEE/CVF International conference on computer vision*, 2019, pp. 6252–6261.

[23] S. Singh, C. Arora, and C. Jawahar, "First person action recognition using deep learned descriptors," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2620–2628.

[24] Y. Tang, Z. Wang, J. Lu, J. Feng, and J. Zhou, "Multi-stream deep neural networks for rgb-d egocentric action recognition," *IEEE Transactions on Circuits and Systems for Video Technology*, vol. 29, no. 10, pp. 3001–3015, 2018.

[25] F. Ragusa, A. Furnari, and G. M. Farinella, "Meccano: A multimodal egocentric dataset for humans behavior understanding in the industrial-like domain," *Computer Vision and Image Understanding (CVIU)*, 2023. [Online]. Available: https://iplab.dmi.unict.it/MECCANO/

[26] X. Wang, T. Kwon, M. Rad, B. Pan, I. Chakraborty, S. Andrist, D. Bohus, A. Feniello, B. Tekin, F. V. Frujeri, N. Joshi, and M. Pollefeys, "Holoassist: an egocentric human interaction dataset for interactive ai assistants in the real world," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, October 2023, pp. 20 270–20 281.

[27] A. Fathi, Y. Li, and J. M. Rehg, "Learning to recognize daily actions using gaze," in *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part I 12*. Springer, 2012, pp. 314–327.

[28] G. A. Sigurdsson, A. Gupta, C. Schmid, A. Farhadi, and K. Alahari, "Actor and observer: Joint modeling of first and third-person videos," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[29] G. A. Sigurdsson, A. Gupta, C. Schmid, A. Farhadi, and K. Alahari, "Charades-ego: A large-scale dataset of paired third and first person videos," in *ArXiv*, 2018.

[30] G. Garcia-Hernando, S. Yuan, S. Baek, and T.-K. Kim, "First-person hand action benchmark with rgb-d videos and 3d hand pose annotations," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 409–419.

[31] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price *et al.*, "Scaling egocentric vision: The epic-kitchens dataset," in *Proceedings of the European conference on computer vision (ECCV)*, 2018, pp. 720–736.

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

[33] T. Kwon, B. Tekin, J. Stühmer, F. Bogo, and M. Pollefeys, "H2o: Two hands manipulating objects for first person interaction recognition," in *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2021, pp. 10 138–10 148.

[34] G. A. Tadesse, O. Bent, K. Weldemariam, M. A. Istiak, T. Hasan, and A. Cavallaro, "Bon: An extended public domain dataset for human activity recognition," *arXiv preprint arXiv:2209.05077*, 2022.

[35] F. Sener, D. Chatterjee, D. Shelepov, K. He, D. Singhania, R. Wang, and A. Yao, "Assembly101: A large-scale multi-view video dataset for understanding procedural activities," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 21 096–21 106.

[36] Y. Liu, Y. Liu, C. Jiang, K. Lyu, W. Wan, H. Shen, B. Liang, Z. Fu, H. Wang, and L. Yi, "Hoi4d: A 4d egocentric dataset for category-level human-object interaction," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2022, pp. 21 013–21 022.

[37] F. Ragusa, A. Furnari, S. Livatino, and G. M. Farinella, "The meccano dataset: Understanding human-object interactions from egocentric videos in an industrial-like domain," in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, January 2021, pp. 1569–1578.

[38] G. Abebe, A. Catala, and A. Cavallaro, "A first-person vision dataset of office activities," in *Multimodal Pattern Recognition of Social Signals in Human-Computer-Interaction: 5th IAPR TC 9 Workshop, MPRSS 2018, Beijing, China, August 20, 2018, Revised Selected Papers 5.* Springer, 2019, pp. 27–37.

[39] I. Radosavovic, R. P. Kosaraju, R. Girshick, K. He, and P. Dollár, "Designing network design spaces," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2020, pp. 10 428–10 436.

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

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

[42] H. Touvron, M. Cord, M. Douze, F. Massa, A. Sablayrolles, and H. Jégou, "Training data-efficient image transformers & distillation through attention," in *International conference on machine learning*. PMLR, 2021, pp. 10 347–10 357.

[43] E. D. Cubuk, B. Zoph, J. Shlens, and Q. V. Le, "Randaugment: Practical automated data augmentation with a reduced search space," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, 2020, pp. 702–703.

[44] H. Zhang, "mixup: Beyond empirical risk minimization," *arXiv preprint arXiv:1710.09412*, 2017.

[45] S. Yun, D. Han, S. J. Oh, S. Chun, J. Choe, and Y. Yoo, "Cutmix: Regularization strategy to train strong classifiers with localizable features," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2019, pp. 6023–6032.

[46] Z. Zhong, L. Zheng, G. Kang, S. Li, and Y. Yang, "Random erasing data augmentation," in *Proceedings of the AAAI conference on artificial intelligence*, vol. 34, no. 07, 2020, pp. 13 001–13 008.

[47] E. Hoffer, T. Ben-Nun, I. Hubara, N. Giladi, T. Hoefler, and D. Soudry, "Augment your batch: Improving generalization through instance repetition," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2020, pp. 8129–8138.

[48] G. Huang, Y. Sun, Z. Liu, D. Sedra, and K. Q. Weinberger, "Deep networks with stochastic depth," in *Computer Vision–ECCV 2016: 14th European Conference, Amsterdam, The Netherlands, October 11–14, 2016, Proceedings, Part IV 14*. Springer, 2016, pp. 646–661.

[49] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, pp. 211–252, 2015.

[50] G. Hinton, O. Vinyals, and J. Dean, "Distilling the knowledge in a neural network," *arXiv preprint arXiv:1503.02531*, 2015.

[51] B. Graham, A. El-Nouby, H. Touvron, P. Stock, A. Joulin, H. Jégou, and M. Douze, "Levit: a vision transformer in convnet's clothing for faster inference," in *Proceedings of the IEEE/CVF international conference on computer vision*, 2021, pp. 12 259–12 269.

[52] S. Ioffe, "Batch normalization: Accelerating deep network training by reducing internal covariate shift," *arXiv preprint arXiv:1502.03167*, 2015.

[53] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting," *The journal of machine learning research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[54] C. Lea, M. D. Flynn, R. Vidal, A. Reiter, and G. D. Hager, "Temporal convolutional networks for action segmentation and detection," in *proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017, pp. 156–165.

[55] S. Bai, J. Z. Kolter, and V. Koltun, "An empirical evaluation of generic convolutional and recurrent networks for sequence modeling," *arXiv preprint arXiv:1803.01271*, 2018.

[56] Y. Du, R. Kips, A. Pumarola, S. Starke, A. Thabet, and A. Sanakoyeu, "Avatars grow legs: Generating smooth human motion from sparse tracking inputs with diffusion model," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 481–490.

[57] A. Van Den Oord, S. Dieleman, H. Zen, K. Simonyan, O. Vinyals, A. Graves, N. Kalchbrenner, A. Senior, K. Kavukcuoglu *et al.*, "Wavenet: A generative model for raw audio," *arXiv preprint arXiv:1609.03499*, vol. 12, 2016.

[58] J. L. Ba, "Layer normalization," *arXiv preprint arXiv:1607.06450*, 2016.

[59] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural networks*, vol. 107, pp. 3–11, 2018.

[60] R. Wightman, "Pytorch image models," https://github.com/rwightman/pytorch-image-models, 2019.

[61] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga *et al.*, "Pytorch: An imperative style, high-performance deep learning library," *Advances in neural information processing systems*, vol. 32, 2019.

[62] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[63] I. Loshchilov, "Decoupled weight decay regularization," *arXiv preprint arXiv:1711.05101*, 2017.

[64] X. Wang, R. Girshick, A. Gupta, and K. He, "Non-local neural networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2018, pp. 7794–7803.

[65] 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.

[66] 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.

[67] B. Tekin, F. Bogo, and M. Pollefeys, "H+ o: Unified egocentric recognition of 3d hand-object poses and interactions," in *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, 2019, pp. 4511–4520.

[68] H. Cho, C. Kim, J. Kim, S. Lee, E. Ismayilzada, and S. Baek, "Transformer-based unified recognition of two hands manipulating objects," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023, pp. 4769–4778.

Appendix A

# Dataset Statistics

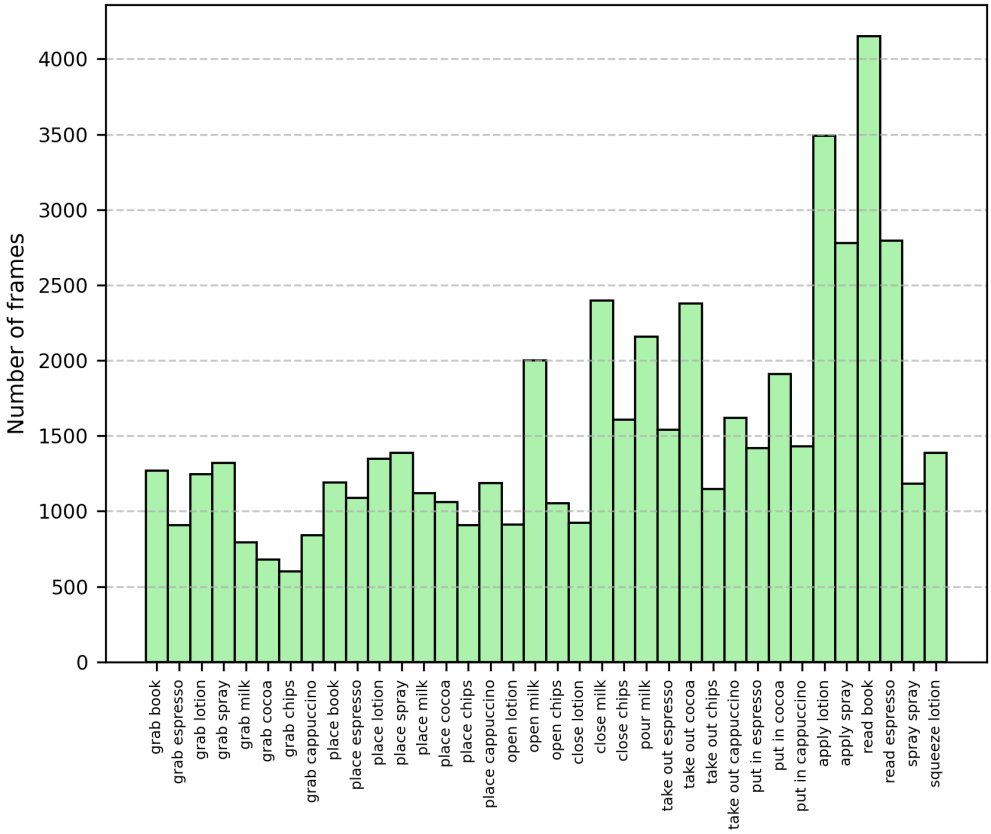


Figure A.1: Number of frames per action class in the training set.

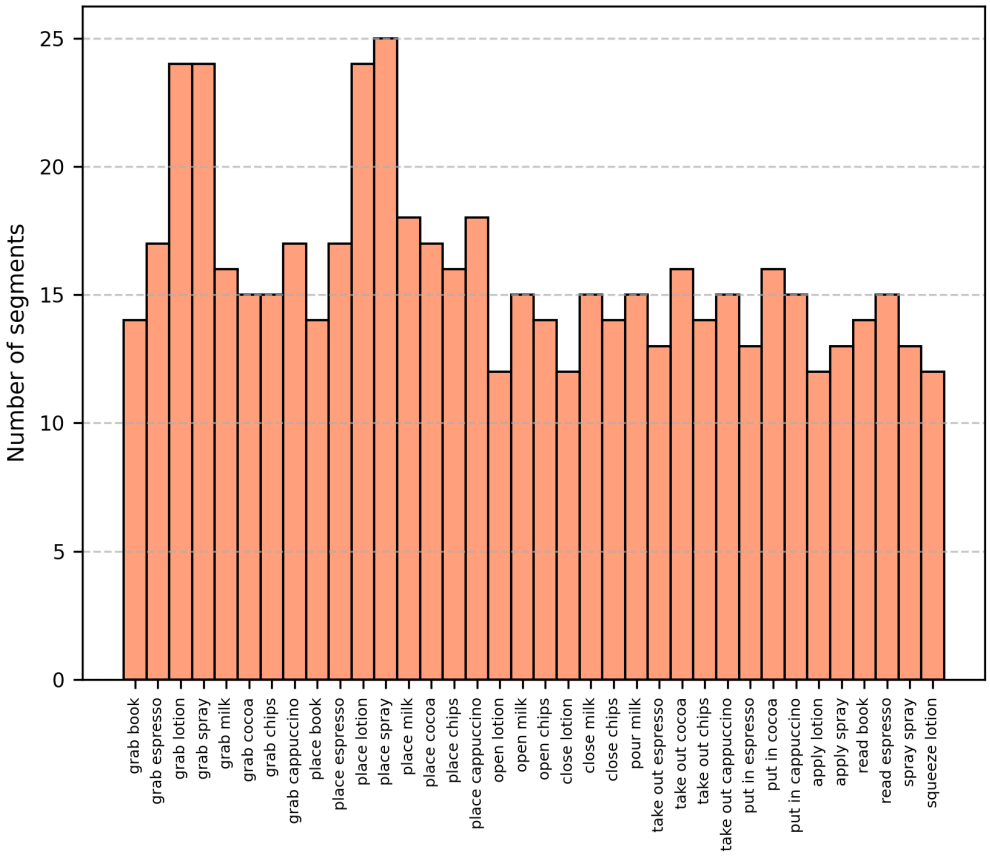


Figure A.2: Number of segments per action class in the training set.

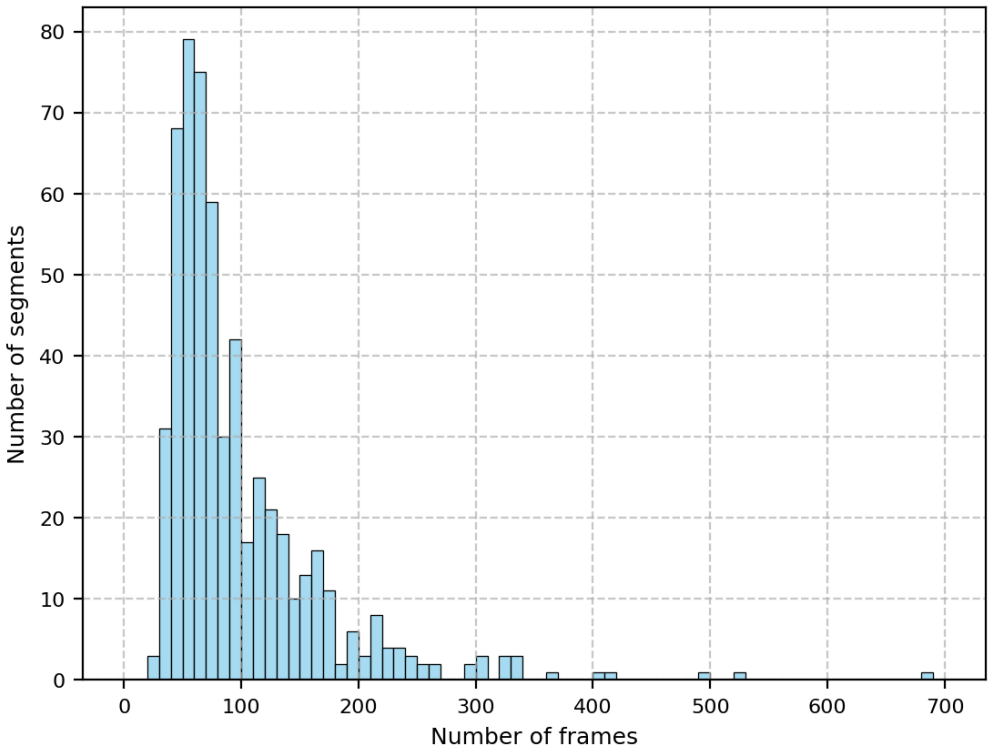


Figure A.3: Distribution of number of segments by number of frames in the training set.

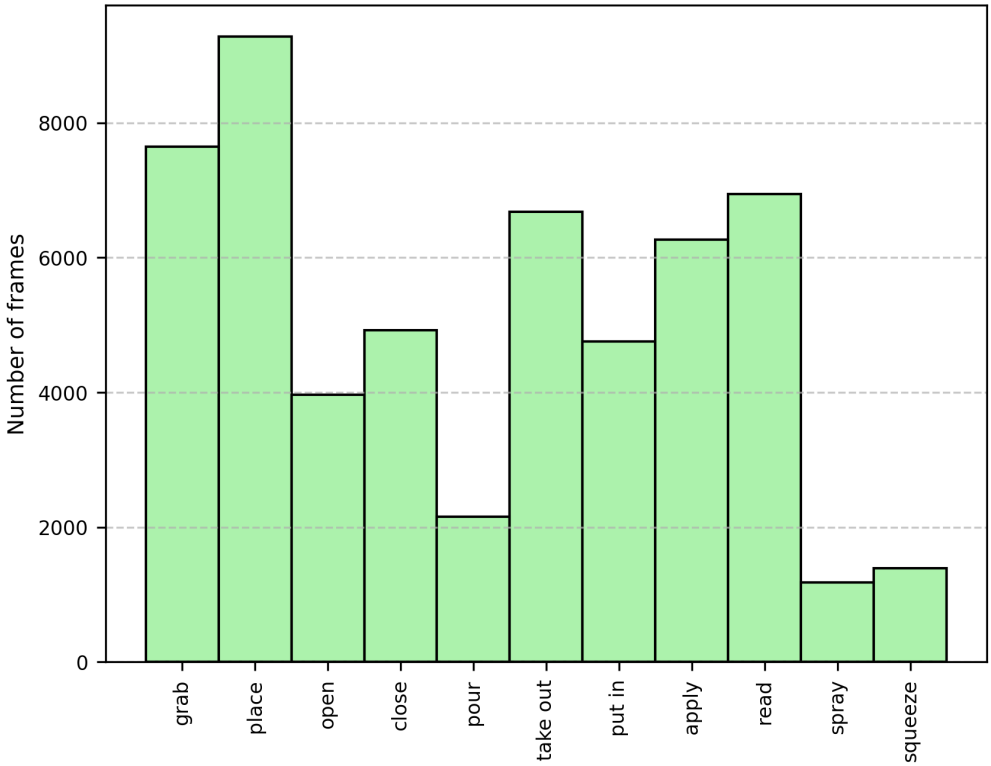


Figure A.4: Number of frames per verb class in the training set.

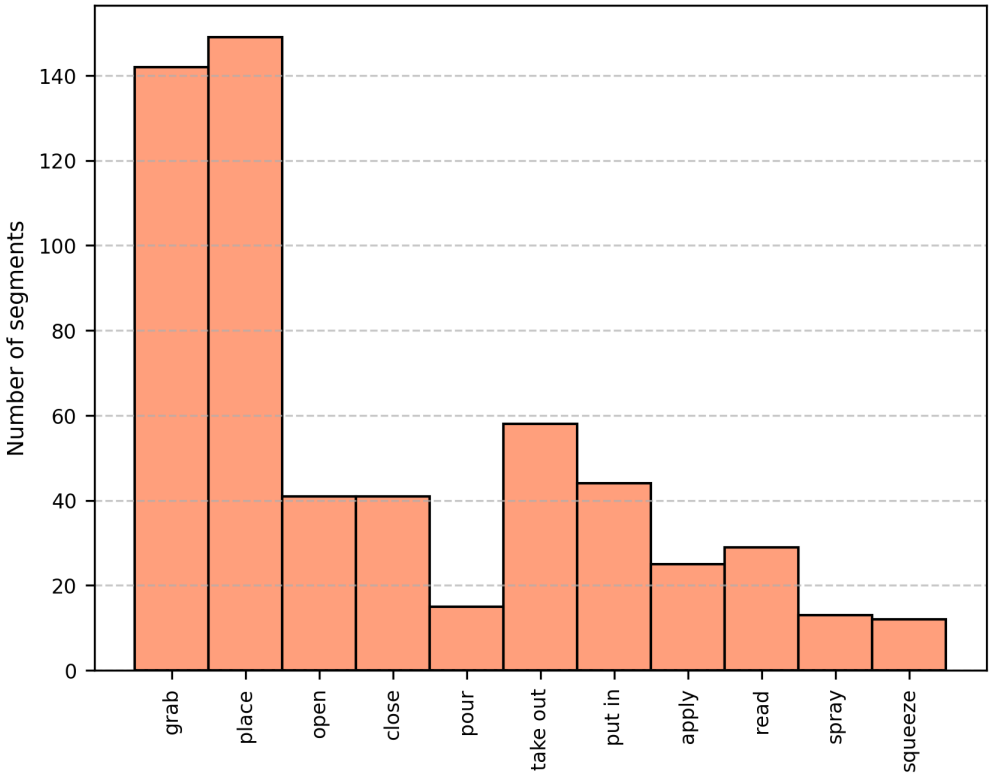


Figure A.5: Number of segments per verb class in the training set.

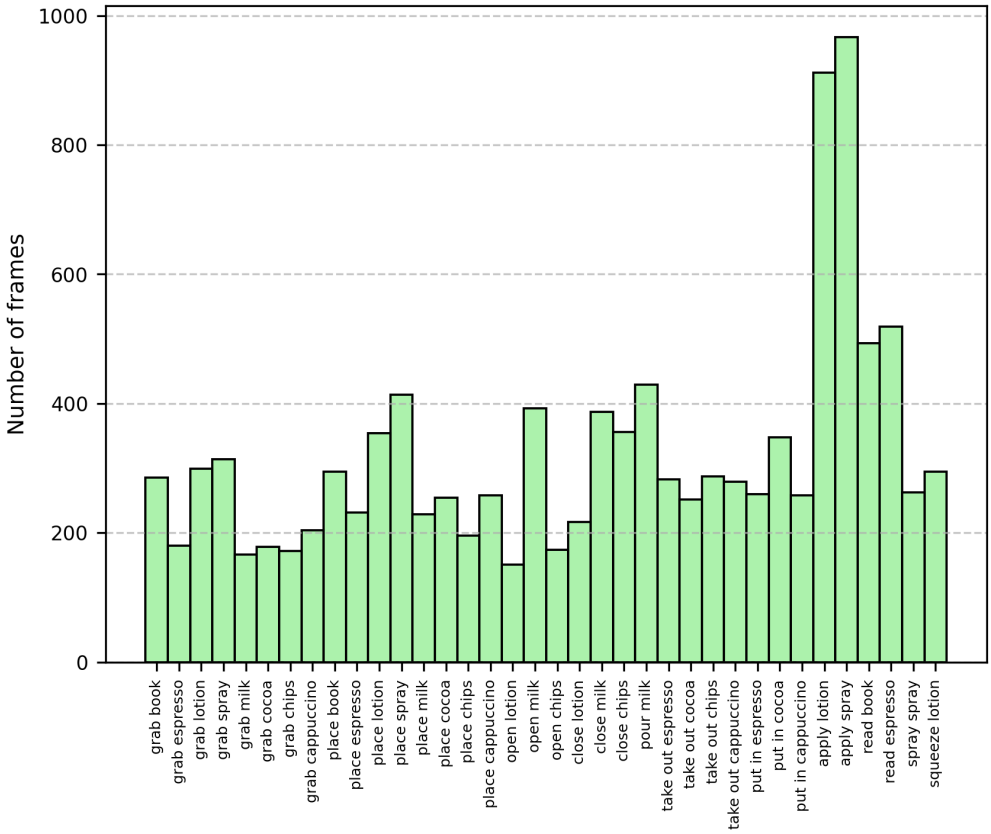


Figure A.6: Number of frames per action class in the evaluation set.

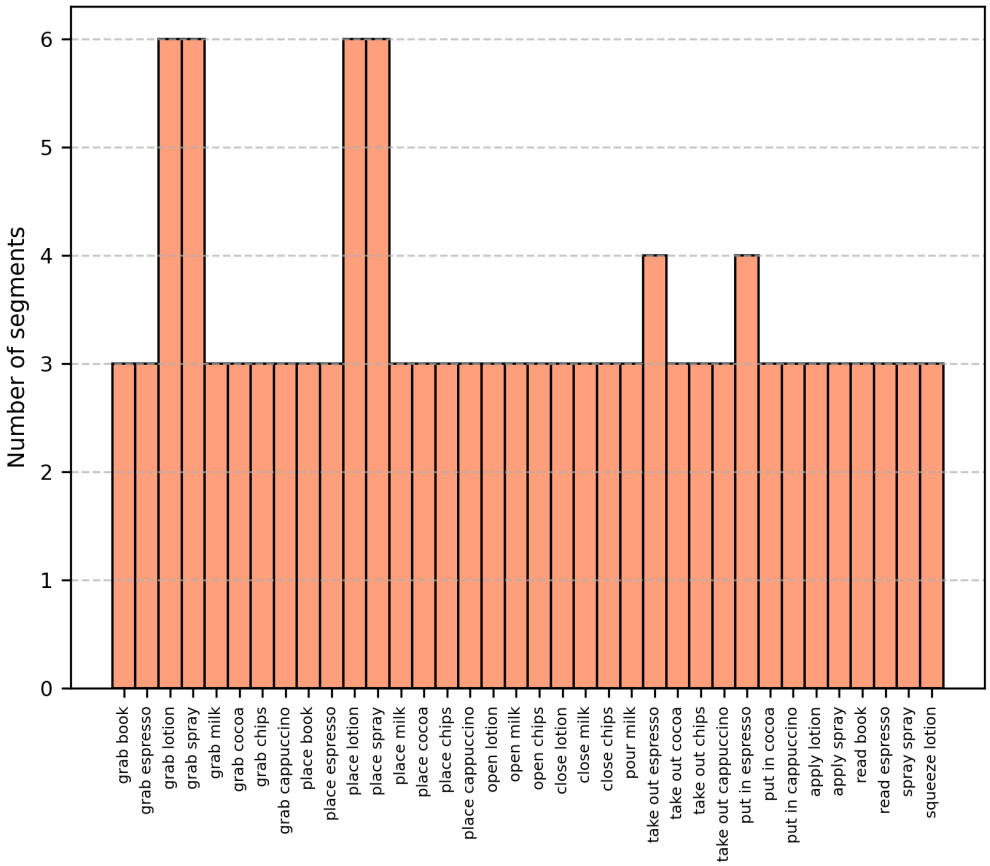


Figure A.7: Number of segments per action class in the evaluation set.

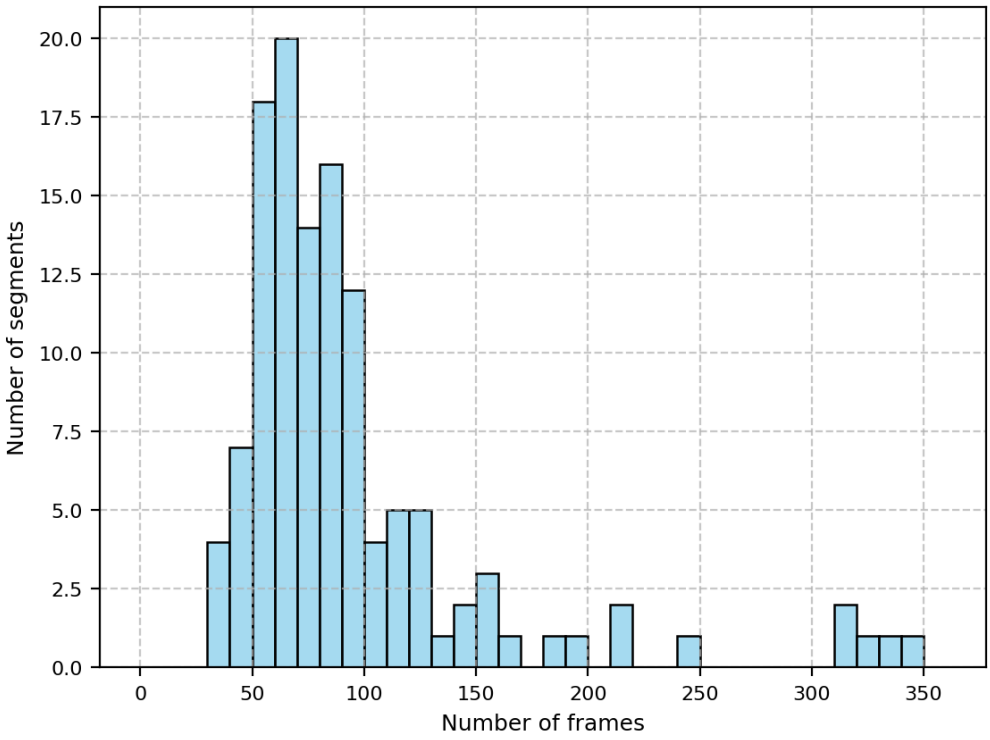


Figure A.8: Distribution of number of segments by number of frames in the evaluation set.

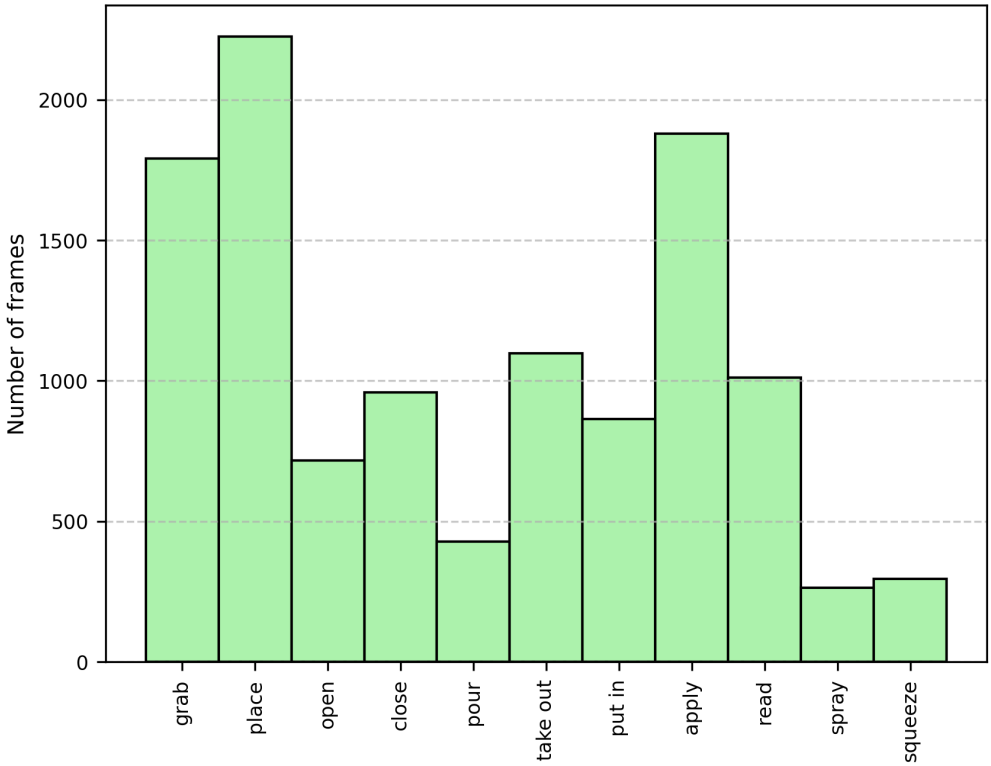


Figure A.9: Number of frames per verb class in the evaluation set.

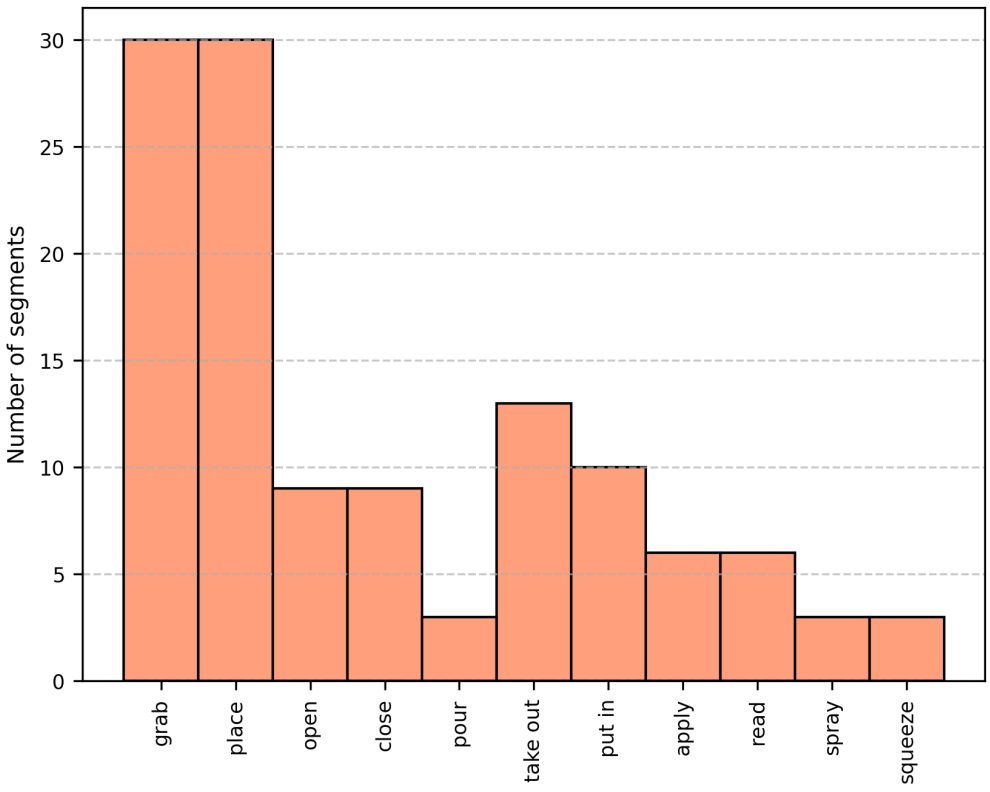


Figure A.10: Number of segments per verb class in the evaluation set.

Appendix B

# Full Results Tables

| Model | Context RGB (s) | Freq RGB (Hz) | Context HP (s) | Freq HP (Hz) | Num Params |
|---|---|---|---|---|---|
| HP-MLP-v | 0 | N/A | 1 | N/A | 0.61M |
| HP-MLP | 0 | N/A | 1 | N/A | 0.62M |
| RegNet-12GF | 1 | N/A | 0 | N/A | 49.66M |
| LeViT-256 | 1 | N/A | 0 | N/A | 17.89M |
| LeViT-256-distilled | 1 | N/A | 0 | N/A | 17.90M |
| FusionNet | 1 | N/A | 1 | N/A | 20.09M |
| TMLP-C2F30C0F0 | 2 | 30 | 0 | 0 | 23.29M |
| TMLP-C2F10C0F0 | 2 | 10 | 0 | 0 | 23.25M |
| TMLP-C2F3C0F0 | 2 | 3 | 0 | 0 | 23.25M |
| TMLP-C2F1C0F0 | 2 | 1 | 0 | 0 | 23.25M |
| TCN-C2F3C0F0 | 2 | 3 | 0 | 0 | 26.63M |
| TCN-C2F1C0F0 | 2 | 1 | 0 | 0 | 26.63M |
| TMLP-C0F0C2F30-v | 0 | 0 | 2 | 30 | 3.79M |
| TMLP-C0F0C2F10-v | 0 | 0 | 2 | 10 | 3.77M |
| TMLP-C0F0C2F3-v | 0 | 0 | 2 | 3 | 3.77M |
| TMLP-C0F0C2F1-v | 0 | 0 | 2 | 1 | 3.77M |
| MM-TMLP-C2F30C2F30 | 2 | 30 | 2 | 30 | 27.49M |
| MM-TMLP-C2F10C2F30 | 2 | 10 | 2 | 30 | 27.45M |
| MM-TMLP-C2F3C2F30 | 2 | 3 | 2 | 30 | 27.44M |
| MM-TMLP-C2F1C2F30 | 2 | 1 | 2 | 30 | 27.44M |
| MM-TMLP-C2F30C2F10 | 2 | 30 | 2 | 10 | 27.47M |
| MM-TMLP-C2F10C2F10 | 2 | 10 | 2 | 10 | 27.43M |
| MM-TMLP-C2F3C2F10 | 2 | 3 | 2 | 10 | 27.42M |
| MM-TMLP-C2F1C2F10 | 2 | 1 | 2 | 10 | 27.42M |
| MM-TMLP-C2F30C2F3 | 2 | 30 | 2 | 3 | 27.46M |
| MM-TMLP-C2F10C2F3 | 2 | 10 | 2 | 3 | 27.43M |
| MM-TMLP-C2F3C2F3 | 2 | 3 | 2 | 3 | 27.42M |
| MM-TMLP-C2F1C2F3 | 2 | 1 | 2 | 3 | 27.42M |
| MM-TMLP-C2F30C2F1 | 2 | 30 | 2 | 1 | 27.46M |
| MM-TMLP-C2F10C2F1 | 2 | 10 | 2 | 1 | 27.42M |
| MM-TMLP-C2F3C2F1 | 2 | 3 | 2 | 1 | 27.42M |
| MM-TMLP-C2F1C2F1 | 2 | 1 | 2 | 1 | 27.42M |

Table B.1: General properties of selected models. Context indicates the length of the context in seconds that the model has access to.

| Model | Frame Accuracy (%) | Frame F1-score (%) | Segment Accuracy (%) | Segment F1-score (%) |
|---|---|---|---|---|
| HP-MLP-v | 78.373 | 79.686 | 80.328 | 85.748 |
| HP-MLP | 52.079 | 46.933 | 53.279 | 51.098 |
| RegNet | 79.997 | 74.844 | 81.967 | 79.437 |
| LeViT-256 | 76.499 | 70.584 | 78.689 | 76.993 |
| LeViT-256-distilled | 79.112 | 73.692 | 81.148 | 79.854 |
| FusionNet | 81.139 | 76.812 | 84.426 | 83.266 |
| TMLP-C2F30C0F0 | 92.327 | 91.154 | 95.082 | 95.352 |
| TMLP-C2F10C0F0 | 90.454 | 88.974 | 90.984 | 91.084 |
| TMLP-C2F3C0F0 | 88.589 | 86.667 | 88.525 | 88.674 |
| TMLP-C2F1C0F0 | 85.900 | 82.809 | 87.705 | 87.322 |
| TCN-C2F3C0F0 | 86.621 | 82.752 | 87.705 | 86.781 |
| TCN-C2F1C0F0 | 85.556 | 82.232 | 89.344 | 89.332 |
| TMLP-C0F0C2F30-v | 86.157 | 85.301 | 87.705 | 88.761 |
| TMLP-C0F0C2F10-v | 86.810 | 86.348 | 87.705 | 88.889 |
| TMLP-C0F0C2F3-v | 85.118 | 84.848 | 84.426 | 85.356 |
| TMLP-C0F0C2F1-v | 84.087 | 82.850 | 83.607 | 85.103 |
| MM-TMLP-C2F30C2F30 | 93.573 | 92.449 | 95.082 | 95.202 |
| MM-TMLP-C2F10C2F30 | 92.421 | 91.677 | 95.082 | 95.228 |
| MM-TMLP-C2F3C2F30 | 89.844 | 88.055 | 90.164 | 90.318 |
| MM-TMLP-C2F1C2F30 | 89.414 | 87.209 | 92.623 | 92.349 |
| MM-TMLP-C2F30C2F10 | 93.487 | 92.527 | 96.721 | 97.107 |
| MM-TMLP-C2F10C2F10 | 92.696 | 91.595 | 94.262 | 94.285 |
| MM-TMLP-C2F3C2F10 | 90.419 | 88.577 | 92.623 | 92.193 |
| MM-TMLP-C2F1C2F10 | 89.457 | 86.540 | 92.623 | 92.349 |
| MM-TMLP-C2F30C2F3 | 93.813 | 92.887 | 95.902 | 96.154 |
| MM-TMLP-C2F10C2F3 | 92.404 | 91.123 | 92.623 | 92.943 |
| MM-TMLP-C2F3C2F3 | 90.084 | 88.499 | 93.443 | 93.324 |
| MM-TMLP-C2F1C2F3 | 88.941 | 86.572 | 91.803 | 91.703 |
| MM-TMLP-C2F30C2F1 | 93.246 | 92.133 | 95.082 | 95.535 |
| MM-TMLP-C2F10C2F1 | 92.026 | 90.520 | 91.803 | 91.991 |
| MM-TMLP-C2F3C2F1 | 88.555 | 86.756 | 88.525 | 88.346 |
| MM-TMLP-C2F1C2F1 | 87.661 | 85.020 | 90.984 | 90.890 |

Table B.2: Accuracy and F1-score of selected model for frame-level and segment-level predictions.

| Model | Time GPU mean (ms) | Time GPU std (ms) | Time CPU mean (ms) | Time CPU std (ms) |
|---|---|---|---|---|
| HP-MLP-v | 0.332 | 0.016 | 0.214 | 0.010 |
| HP-MLP | 0.330 | 0.007 | 0.215 | 0.003 |
| RegNet-12GF | 8.961 | 0.052 | 336.472 | 2.098 |
| LeViT-256 | 8.615 | 0.189 | 45.988 | 1.244 |
| LeViT-256-distilled | 8.571 | 0.157 | 44.986 | 0.691 |
| FusionNet | 9.223 | 0.231 | 46.331 | 0.564 |
| TMLP-C2F30C0F0 | 22.028 | 0.203 | 2320.444 | 6.069 |
| TMLP-C2F10C0F0 | 16.200 | 0.279 | 720.304 | 2.470 |
| TMLP-C2F3C0F0 | 16.072 | 0.272 | 219.823 | 1.272 |
| TMLP-C2F1C0F0 | 16.180 | 0.268 | 82.654 | 0.653 |
| TCN-C2F3C0F0 | 13.682 | 0.241 | 251.176 | 1.950 |
| TCN-C2F1C0F0 | 13.652 | 0.192 | 100.226 | 0.742 |
| TMLP-C0F0C2F30-v | 4.118 | 0.017 | 10.455 | 0.309 |
| TMLP-C0F0C2F10-v | 4.089 | 0.027 | 5.428 | 0.064 |
| TMLP-C0F0C2F3-v | 4.002 | 0.021 | 3.938 | 0.294 |
| TMLP-C0F0C2F1-v | 4.024 | 0.027 | 3.136 | 0.036 |
| MM-TMLP-C2F30C2F30 | 25.891 | 0.246 | 2306.591 | 3.887 |
| MM-TMLP-C2F10C2F30 | 20.330 | 0.274 | 739.656 | 2.359 |
| MM-TMLP-C2F3C2F30 | 20.245 | 0.285 | 229.074 | 1.265 |
| MM-TMLP-C2F1C2F30 | 20.377 | 0.271 | 93.523 | 0.562 |
| MM-TMLP-C2F30C2F10 | 25.832 | 0.305 | 2292.625 | 3.846 |
| MM-TMLP-C2F10C2F10 | 20.305 | 0.271 | 722.962 | 2.351 |
| MM-TMLP-C2F3C2F10 | 20.222 | 0.285 | 224.186 | 1.283 |
| MM-TMLP-C2F1C2F10 | 20.323 | 0.267 | 88.688 | 0.684 |
| MM-TMLP-C2F30C2F3 | 25.799 | 0.228 | 2291.908 | 3.996 |
| MM-TMLP-C2F10C2F3 | 20.231 | 0.273 | 733.177 | 2.277 |
| MM-TMLP-C2F3C2F3 | 20.125 | 0.267 | 223.906 | 1.266 |
| MM-TMLP-C2F1C2F3 | 20.245 | 0.267 | 87.103 | 0.742 |
| MM-TMLP-C2F30C2F1 | 25.780 | 0.240 | 2301.329 | 4.333 |
| MM-TMLP-C2F10C2F1 | 20.277 | 0.275 | 724.087 | 2.416 |
| MM-TMLP-C2F3C2F1 | 20.162 | 0.268 | 222.389 | 1.256 |
| MM-TMLP-C2F1C2F1 | 20.318 | 0.268 | 86.358 | 0.743 |

Table B.3: Inference time of selected models on GPU and CPU.

| Model | Time CPU mean (ms) | Time CPU std (ms) | Usage (%) |
|---|---|---|---|
| TMLP-C2F30C0F0 | 1832.083 | 12.922 | 183.208 |
| TMLP-C2F10C0F0 | 537.309 | 3.364 | 53.731 |
| TMLP-C2F3C0F0 | 154.724 | 1.670 | 15.472 |
| TMLP-C2F1C0F0 | 50.596 | 0.510 | 5.060 |
| TCN-C2F3C0F0 | 906.011 | 2.716 | 90.691 |
| TCN-C2F1C0F0 | 725.977 | 1.348 | 72.598 |
| TMLP-C0F0C2F30-v | 282.789 | 1.432 | 28.279 |
| TMLP-C0F0C2F10-v | 52.036 | 0.148 | 5.204 |
| TMLP-C0F0C2F3-v | 11.330 | 0.072 | 1.133 |
| TMLP-C0F0C2F1-v | 3.207 | 0.036 | 0.321 |
| MM-TMLP-C2F30C2F30 | 2118.842 | 1.193 | 211.884 |
| MM-TMLP-C2F10C2F30 | 823.950 | 0.859 | 82.395 |
| MM-TMLP-C2F3C2F30 | 444.983 | 0.549 | 44.498 |
| MM-TMLP-C2F1C2F30 | 339.411 | 0.748 | 33.941 |
| MM-TMLP-C2F30C2F10 | 1880.565 | 1.001 | 188.056 |
| MM-TMLP-C2F10C2F10 | 588.946 | 0.769 | 58.895 |
| MM-TMLP-C2F3C2F10 | 207.478 | 0.337 | 20.748 |
| MM-TMLP-C2F1C2F10 | 103.454 | 0.202 | 10.345 |
| MM-TMLP-C2F30C2F3 | 1842.905 | 0.900 | 184.290 |
| MM-TMLP-C2F10C2F3 | 548.006 | 0.617 | 54.801 |
| MM-TMLP-C2F3C2F3 | 165.934 | 0.317 | 16.593 |
| MM-TMLP-C2F1C2F3 | 62.461 | 0.146 | 6.246 |
| MM-TMLP-C2F30C2F1 | 1833.635 | 0.957 | 183.363 |
| MM-TMLP-C2F10C2F1 | 540.224 | 0.624 | 54.022 |
| MM-TMLP-C2F3C2F1 | 157.785 | 0.282 | 15.778 |
| MM-TMLP-C2F1C2F1 | 53.618 | 0.176 | 5.362 |

Table B.4: CPU usage of selected models.