Università degli Studi di Padova

Dipartimento di Fisica e Astronomia Galileo Galilei

corso di laurea magistrale in Physics of Data

Sarvenaz Babakhani

Human action detection in temporal data

elaborato finale

Relatori:
Prof. Marco Zanetti (internal Supervisor)
Prof. Alina Roitberg(external Supervisor)

---

anno accademico 2023-2024

Indice

Abstract

This thesis addresses the challenge of estimating caloric expenditure from videos capturing individuals engaging in a variety of activities, ranging from mild to intense exercises. The estimation of calories burned is based not only on the category of physical activity (e.g., running, walking) but also on assessing the intensity of muscle and bodily movements depicted in the videos.

To approach this goal, tests were conducted on two distinct sets of data: "known test", which includes activities represented in the training dataset, and "unknown test", consisting of activities not featured during training. This methodology allowed the exploration of not only the model's performance on familiar activities but also its generalization capabilities to new, unseen actions, to check the effect of categories on the training.

In this study, neural networks are leveraged, specifically utilizing foundation models due to computational constraints. A pre-trained Dino model is used to extract features from video data. These features are then fed into an evaluator model to estimate caloric burn.

A significant portion of this research involved experimenting with different loss functions and tuning various parameters to optimize the model's predictive accuracy. Through these experiments, the aim was to enhance the model's ability to accurately estimate calorie expenditure, thereby contributing valuable insights into the domain of human action detection in temporal data.

Capitolo 1

Introduction

Human action detection and caloric expenditure estimation are important areas of research within computer vision and health monitoring. Traditional methods for caloric estimation often rely on physiological measurements and wearable sensors, which can be impractical for everyday use. With the advancement of deep learning, especially in video analysis, there is an opportunity to use video data to estimate energy expenditure more accurately.

This research wants to bridge the gap between action recognition and caloric estimation by utilizing advanced neural network architectures, specifically foundation models like Dino and CLIP. These models are pre-trained on extensive datasets, allowing them to learn generalizable features. By integrating these foundation models with task-specific neural networks and exploring different loss functions, the study seeks to develop an end-to-end pipeline that can accurately detect human actions and estimate caloric burn from video data.

The introduction of large-scale dataset which includes diverse activities with detailed annotations, provides a good foundation for training and evaluating these models. This dataset, combined with the advanced capabilities of foundation models, aims to overcome the limitations of existing methods, such as rigid categorization and poor generalization, thereby enhancing the practical applicability of caloric estimation systems in real-world scenarios.

Overall, this research not only addresses the technical challenges in video-based caloric estimation but also opens a new window for applications in mobile health, fitness tracking, and beyond. By advancing the understanding and implementation of deep learning models in this domain, the study contributes valuable

insights and tools for more effective health monitoring and activity analysis.

1.1 Review of related research

1.1.1 Human Action Detection

Human action detection has been a significant area of research within computer vision, primarily focusing on recognizing and classifying activities in videos. The capability to recognize, interpret, and predict complex human actions helps to develop critical applications, including intelligent surveillance systems, human-computer interfaces, and health care.[1] [2] [3]

Early approaches relied heavily on handcrafted features and traditional machine-learning algorithms. But, in recent years, deep learning, particularly convolutional neural networks (CNNs) [4] [5] [6] and recurrent neural networks (RNNs)[7], has revolutionized human action detection. CNNs, such as the 3D ConvNet, extend the 2D convolutions to the temporal dimension, capturing both spatial and temporal features simultaneously. RNNs, especially Long Short-Term Memory (LSTM) networks [8], have been utilized to model temporal dependencies in sequential data, enhancing the performance of action detection systems.

Notable advancements include the development of two-stream networks that process spatial and temporal information separately and then combine them to improve accuracy. The introduction of attention mechanisms [9] and transformers [10] has further pushed the boundaries, allowing models to focus on relevant parts of the video frames and capture long-range dependencies more effectively.

The research in this field has been accelerated by the availability of large-scale activity recognition datasets from sources such as YouTube, movies, and controlled home environments. These datasets have fueled advancements in applications ranging from cooking and sports to robotics and automated driving. Datasets such as Kinetics [11] and ActivityNet[12] provide a diverse range of activities captured in varied environments, contributing to more robust and generalizable models and for better results, researches have also explored the uncertainty of video classification models, addressing the challenges of ambiguous or overlapping action classes[13].
Despite these strides, applying these sophisticated techniques to estimate complex physiological processes, such as caloric expenditure, remains an emerging

and relatively unexplored area[14]. Future research needs to focus on bridging this gap, leveraging the strengths of current methodologies to develop models capable of accurately estimating energy expenditure from human actions captured in video data.

These advancements highlight the ongoing evolution in human action detection, where integrating cutting-edge neural network architectures with extensive datasets paves the way for more accurate and efficient applications across various domains. The continued exploration of these techniques will likely yield significant contributions to fields such as health monitoring, sports analytics, and intelligent surveillance systems.

1.1.2 Caloric Expenditure Estimation

Estimating caloric expenditure from physical activities has traditionally relied on physiological measurements such as heart rate, oxygen consumption, and metabolic equivalents (METs). Wearable devices equipped with accelerometers and gyroscopes are commonly used to monitor these metrics and estimate energy expenditure [15][16][17][18]. Another method is the Optical flow method which tracks the movement of objects within video frames, providing a dense motion field that can be used to estimate the intensity of physical activities. By analyzing the motion vectors, models can infer the speed and direction of movements, which are critical indicators of energy expenditure [19][20]. With the improvement of computer vision, researchers have found video-based caloric expenditure estimation, to infer energy expenditure directly from visual data. This approach uses pose estimation [21] and action recognition to identify and quantify physical activities in videos. Models such as OpenPose [22] and AlphaPose[23] have tried to enable machines to extract detailed human pose information, which serves as a foundation for estimating the intensity and type of physical activities. Beyond traditional pose estimation, skeleton-based action recognition leverages the spatial and temporal dynamics of human skeleton joints to classify actions and estimate energy expenditure. Methods like ST-GCN (Spatial Temporal Graph Convolutional Networks) have shown promise in capturing the nuanced movements of different activities by modeling the joint connections and their movements over time [24].

Recent studies have integrated deep learning techniques to improve the accuracy of caloric expenditure estimation. These models often combine pose estimation with activity classification, using features extracted from video frames to predict the calories burned during different activities. However, challenges remain in accurately modeling the relationship between visual features and energy expenditure, particularly for activities involving complex or subtle movements.

1.1.3 Use of Neural Networks in Video Analysis

Neural networks have become a cornerstone of video analysis, driven by their ability to automatically learn and extract relevant features from raw data. In the context of human action detection and caloric expenditure estimation, several neural network architectures have been employed to address various challenges.[25][26]

Pretrained models, such as self-supervised deep learning, a form of self-distillation with no labels (Dino), and Contrastive Language-Image Pre-training (CLIP), have demonstrated remarkable capabilities in feature extraction from video data. These models are trained on large-scale datasets and can transfer learned representations to new tasks with limited additional training, making them suitable for applications with computational constraints.
Dino's self-supervised learning approach makes it particularly useful for tasks like human action recognition. By capturing detailed spatial and temporal features from video frames, Dino can effectively model the complexities of human movements. Its ability to generate high-quality feature representations without requiring labeled data makes it a powerful tool for video analysis tasks [27].
CLIP, aligns visual and textual representations through contrastive learning. CLIP learns to understand images contextually, making it highly adaptable for various visual tasks, including action recognition and caloric estimation. This model can distinguish subtle differences in activities by using its understanding of visual semantics. By training on a diverse range of images and their associated textual descriptions, CLIP can effectively generalize to various visual tasks, including action recognition and caloric expenditure estimation[28].

The integration of these foundation models with task-specific neural networks allows for end-to-end learning pipelines that can accurately detect human actions and estimate caloric expenditure from video data. This approach not only leverages the strengths of pre-trained models but also adapts them to the unique

requirements of the target application, enhancing performance and efficiency. By integrating foundation models with neural networks designed for specific tasks, such as CNNs for spatial feature extraction and RNNs or LSTMs for temporal sequence modeling, the pipeline can process video data holistically. This integration enables the system to recognize complex patterns in human movements and accurately estimate the corresponding caloric expenditure.

Task-specific neural networks can be optimized using specialized loss functions that are more suitable for caloric estimation and action recognition. For instance, combining classification loss with regression loss helps in precisely estimating the energy expenditure associated with different actions.

1.1.4 Foundation models

Foundation models are large-scale pre-trained neural networks that serve as a starting point for various downstream tasks. These models are trained on extensive datasets and can be fine-tuned for specific applications, significantly reducing the need for task-specific data and computational resources[29]. Foundation models exhibit exceptional capabilities in understanding, generating, and adapting content across a wide range of domains, including creative generation[30].

- Dino (Self-Supervised Learning with Vision Transformers):

Dino (self-distillation with no labels) employs vision transformers to learn rich visual representations from unlabeled data. Its self-supervised learning framework captures detailed spatial and temporal features, making it highly effective for tasks like human action recognition and caloric expenditure estimation. Dino's ability to model both spatial configurations and temporal dynamics enhances the accuracy of detecting complex human actions and estimating caloric expenditure from videos. DINOv2 builds upon the foundations laid by DINO, introducing improvements in training methods, model architecture, and data efficiency.

In this model, two key points are Vision Transformers which Capture fine-grained visual details and relationships, and Self-Supervised Learning which Utilizes large-scale, unlabeled data for robust feature learning[27][31].

- CLIP (Contrastive Language-Image Pre-training):

CLIP, developed by OpenAI, aligns visual and textual representations through contrastive learning. It is a neural network that learns visual concepts from natural language supervision, allowing it to be applied to any visual classification benchmark using only the names of the visual categories. Unlike traditional models, CLIP leverages a variety of images and natural language available online, enabling it to perform diverse classification tasks without direct benchmark optimization. This approach significantly improves robustness and adaptability, achieving strong performance on benchmarks like ImageNet zero-shot, without relying on extensive labeled datasets, thus addressing key limitations in current computer vision methods. Scaling a simple pre-training task can achieve competitive zero-shot performance on various image classification datasets. CLIP uses text paired with images from the internet to predict which text snippet matches a given image. This method requires CLIP to learn to recognize and associate a wide range of visual concepts with their names. Consequently, CLIP can be applied to diverse classification tasks by predicting the most likely text description paired with each image, such as distinguishing between photos of dogs and cats.

By training on diverse datasets of images and text, CLIP is good at text understanding, making it adaptable to visual tasks, including action recognition. CLIP's robust understanding of visual semantics allows it to differentiate subtle actions in videos, crucial for accurate energy expenditure predictions[28].

1.2 The purpose of thesis

When estimating caloric expenditure from human observations, traditional methods face two significant challenges. First, these methods often rely on rigid categorization into predefined actions, which are typically coarse and context-dependent, such as distinguishing between "running" and "boxing." This categorization provides an easy shortcut for the network but fails to capture the detailed movements that obtain energy expenditure, as medical research highlights the importance of muscle activity and intensity.

Second, deep neural networks tend to learn shortcuts, such as memorizing average values for specific activity categories seen during training, instead of understanding the underlying factors of energy expenditure. Although annotations might be continuous calorie values, the training set can only cover a finite number of activity types. Ideally, a model should understand the nature

of activity-induced energy expenditure by analyzing the type and intensity of bodily movements rather than relying on category-specific biases.

Furthermore, the lack of diverse and comprehensive datasets for training these models exacerbates the problem. Many existing datasets are limited in scope, focusing on specific activities or environments, which hinders the models' ability to generalize across different contexts. To address this challenge, the dataset we use is designed for estimating caloric expenditure from videos. This dataset includes video examples with annotations based on medical models, considering current activity categories, skeleton movement intensity, and heart rate measurements. The dataset includes diverse activities from YouTube, movies, and household contexts. It also contains a cross-category part, evaluating models on activity types not seen during training to ensure generalization [14].

To achieve accurate energy expenditure estimation, the study utilizes foundation models and explores various loss functions. This approach aims to develop models that understand the intensity and type of body movements, rather than relying on category-specific biases.

This thesis emphasizes the need for further research to develop models capable of fine-grained movement analysis and robust generalization to new activities. This work highlights the potential for practical applications in mobile health and the necessity of addressing the key technical challenges in video-based caloric expenditure estimation.

The primary objective of this study is to develop an advanced, accurate, and generalizable model for estimating caloric expenditure from video data. The specific objectives include:

- Leverage Foundation Models: Utilize pre-trained foundation models like Dino and CLIP to extract robust and generalizable features from video data, thereby enhancing the initial stages of the analysis.
- Integrate Task-Specific Neural Networks: Combine foundation models with task-specific neural networks to create an end-to-end pipeline capable of accurately detecting human actions and estimating caloric expenditure.
- Explore Various Loss Functions: Experiment with different loss functions to optimize the model's performance, ensuring it can accurately capture the nuances of human movements and their associated energy expenditure.

In this thesis, We want to study if the Foundation models can improve the feature extraction process, leading to higher accuracy in both action recognition

and caloric expenditure estimation. We also want to compare the effect of some loss functions on optimizing model performance and finally see the result of the evaluation of the network on unseen activities during training.

Capitolo 2

Methodology

2.1 Data

2.1.1 Datasets Overview

This study utilizes a subset of the Vid2Burn-Diverse dataset, which includes around 3,000 video examples. Vid2Burn (Diverse and ADL) is designed by Peng et al. [14] to capture a wide range of activities with detailed annotations and the main resources used in the paper are UCF-101 [32], HMDB51 [33], and the test set of Kinetics [11]. The dataset is divided into "known test" and "unknown test" sets. The "known test" set comprises activities that are present in the training data, ensuring that the model has previously encountered these activities. In contrast, the "unknown test" set includes activities not seen during training, providing a robust evaluation of the model's ability to generalize to new, unseen actions.

2.1.2 Data Collection and Annotation

Data Collection
The videos in the Vid2Burn-Diverse subset were sourced from a variety of activity recognition datasets, including those originally compiled from YouTube and movies. This diverse collection approach ensures a wide coverage of different human activities, contributing to the robustness and comprehensiveness of the dataset.

The collection process involved selecting videos that represent a broad spectrum of physical activities, from low-intensity movements to high-intensity exercises. This diversity is crucial for developing models that can accurately estimate caloric expenditure across different types of activities.

Annotation Process

Based on the paper [14] the annotation of the Vid2Burn-Diverse subset focused on two primary aspects:

Current Activity Category:

Each video is labeled with the specific activity being performed. Categories include common activities such as walking, running, cooking, cleaning, and other household tasks. This categorization helps in identifying the type of activity, which is a critical factor in estimating energy expenditure. Then they [14] use activity-specific metabolic rate values from published compendium by [34]. The annotations were derived based on established medical models, which consider the type and intensity of activities as the main drivers of energy expenditure. This approach ensures that the dataset is both accurate and relevant to the study's objectives.

Intensity of Skeleton Movement:

The same type of activity can be done in different ways since the amount of calories burned is directly related to the amount of active muscles and their intensity. The intensity of movements is quantified by analyzing skeletal data extracted from the videos. As the paper [14] explains, this involves using pose estimation techniques to track key points on the body, such as joints and limbs, to measure the power and extent of movements. These measurements are essential for determining the energy expenditure associated with different activities, as they provide insights into how hard the muscles are working. They estimate the skeleton movement using AlphaPose [35],[36],[37] for Vid2Burn-Diverse. Then they use the model of Tsou et al [38] to approximate the caloric cost induced by the movement.

The category-level values are primarily sourced from well-studied and easily accessible published averages for specific categories. These values provide a reliable basis for initial estimates. They then correct the estimations for individual videos using a detailed body-movement model, which results in more accurate and precise sample-level annotations.

The dataset also incorporates a cross-category benchmark, where the caloric

cost estimation models are evaluated against activities not seen during training. This benchmark is critical for assessing the models' generalization capabilities and ensuring that they can accurately estimate caloric expenditure in real-world scenarios, where activities may vary widely.

By utilizing the Vid2Burn-Diverse dataset with detailed annotations, this study aims to develop and validate models that can accurately detect human actions and estimate the corresponding caloric expenditure, addressing significant gaps in the current literature and advancing the field of video-based health monitoring.

2.1.3 Data Statistics

To promote the task of visually estimating the hourly amount of kilocalories burned during various activities, this study uses a subset of 3,400 videos from the Vid2Burn-Diverse dataset. The dataset includes videos covering a range of activities, with continuous calorie values between 0 to 1000 kcal per hour.

Dataset Splits: The dataset is divided into three subsets:

Training Set: 2,476 videos across 27 activity categories, with an average of 91 videos per category.

The most frequent activity is running, with 169 videos, and the least frequent is skydiving, with 65 videos. The highest average caloric expenditure is for the punching category (around 886 kcal/hour), and the lowest is for sitting (around 89 kcal/hour).

Known Test Set: 640 videos from the same 27 categories used in the training set.

Cross Test (unknown) Set: 284 videos from six unique categories not seen during training.

The splits are designed to evaluate the model's performance on both familiar activities and new, unseen activities, ensuring robustness and generalization capabilities.

2.1.4 Visualizations

Three plots are provided to illustrate the dataset's characteristics:

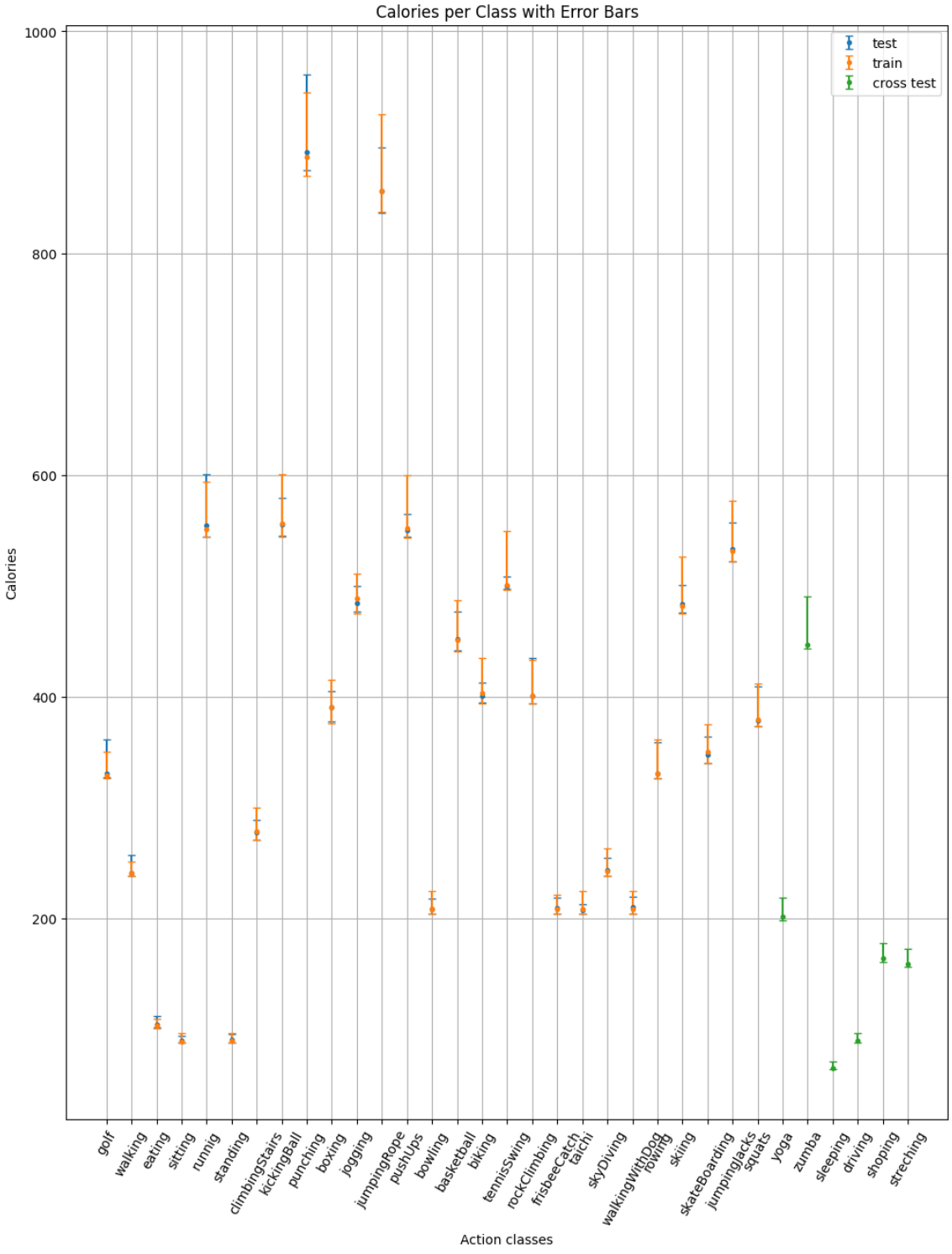

Figura 2.1: Calories per Class with Error Bars: A bar chart with error bars showing the average caloric expenditure per activity class, differentiating between training, test, and cross-test sets.

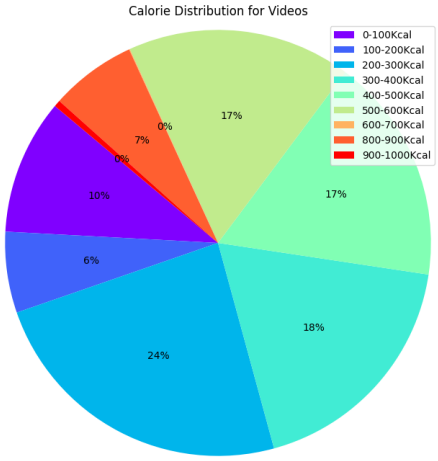

Figura 2.2: Calorie Distribution for Videos: A pie chart showing the distribution of videos across different caloric expenditure ranges.

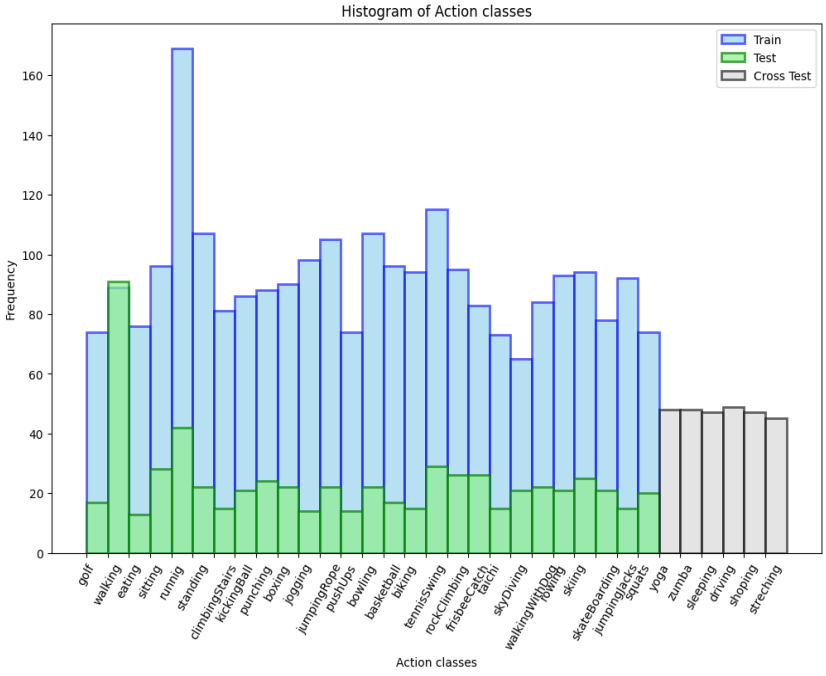

Figura 2.3: Histogram of Action Classes: A histogram showing the frequency of each action class across the training, test, and cross-test sets.

2.1.5 Data Preprocessing

This study's preprocessing steps are essential to prepare the video data for feature extraction and subsequent analysis. The following preprocessing pipeline is employed to ensure consistency and suitability for input into the models:

Loading Video Frames:

Videos are read frame by frame using OpenCV's cv2.VideoCapture. We choose 90 frames of each video to ensure consistent input sizes for the model. If the total number of frames in the video is greater than the required number of frames (90 in this case), we choose 90 frames of the total number of frames, by a random start point. If the video has fewer frames than required, all frames are selected, and then the existing frames are duplicated until the length reaches 90.

Resizing and Cropping:

A transformation pipeline is defined using torchvision.transforms to resize and crop the video frames.
T.Resize(256, interpolation=3, antialias=True): Resizes the shorter side of the image to 256 pixels while maintaining the aspect ratio. The interpolation method used is bicubic (3).
T.CenterCrop(224): Crops the center of the image to a size of 224x224 pixels.

Normalization:
T.ToTensor(): Converts the image to a PyTorch tensor and scales the pixel values to the range [0, 1].
T.Normalize((0.485, 0.456, 0.406), (0.229, 0.224, 0.225)) for DINOv2 and
T.Normalize((0.481, 0.457, 0.408),([0.268, 0.261, 0.275)) for CLIP: Normalizes the tensor using the mean and standard deviation values required by models.

The preprocessing steps ensure that the input size is consistent with the requirements of the DINOv2 and CLIP model and the normalization is crucial as it standardizes the input data, facilitating better performance of the neural network.

In this stage, data are ready to be fed to foundation models to get the features of the videos. For DINOv2, we feed each 16-frame segment with overlap. However, for the CLIP model, the entire video is fed directly into the model, as it handles frame extraction internally. This ensures that the preprocessing pipeline

is tailored to the specific requirements of each model, facilitating efficient and effective feature extraction.

After extracting the features, they should be normalized, and the annotations will be assigned to them to be ready for the calorie evaluator's next training step. This ensures that the data is prepared appropriately for subsequent stages.

2.1.6 Augmentation

Feature augmentation can be a useful technique to improve the robustness and generalization of your model, especially when dealing with limited data or facing overfit. Feature augmentation involves applying transformations directly to the features rather than the raw data. This can be particularly useful in scenarios where the raw data (e.g., video frames) is expensive to process or when working with precomputed features from a pre-trained model [39][40]. There are some techniques for augmenting features, we used in this study:

1. Add Gaussian Noise:
   Gaussian noise is produced by generating a random normal distribution with a mean of 0 and a standard deviation of 0.1 for each feature tensor of the same size, which is then summed with the original feature tensor. This augmentation technique ensures that the model learns to handle slight variations in the data, improving its generalization capabilities.

2. Feature Dropout:
   Randomly zeroing out some elements of the feature vector, similar to dropout applied in neural networks, can also help prevent overfitting and make the model more robust. We generate a mask with the same shape as the feature tensor, where each element has a probability of being zeroed out based on the specified dropout rate. The feature tensor is then element-wise multiplied by this mask. Applying these feature augmentation techniques allows the model to better generalize to unseen data, improving its robustness and overall performance in estimating caloric expenditure from video data.

3. Feature Scaling:
   Randomly scaling the features can also act as a form of augmentation.It is done by generating a random scale factor within a range of (0.9,1.1) for each feature tensor and multiplying the tensor by this scale factor. This

augmentation technique introduces variability in the feature magnitudes, helping the model to generalize better.

4. Random Shifts:
   Randomly shifting the features along the temporal dimension for sequence data can provide temporal robustness. we generate a random integer shift value between -2 and 2. This determines how many steps the features will be shifted. If the shift value is positive, the function pads the features at the beginning (pre-pending zeros) and removes the same number of elements from the end. This effectively shifts the sequence forward. If the shift value is negative, the function pads the features at the end (appending zeros) and removes the same number of elements from the beginning. This shifts the sequence backward. If the shift value is zero, the features remain unchanged. This technique helps the model to be robust to slight misalignments in the temporal sequence of the data.

Applying these augmentation techniques makes the video dataset more diverse, helping the model learn to handle various real-world scenarios and reducing the risk of overfitting. This approach ensures that the trained model performs well on unseen data, making it robust and reliable.

Video augmentation techniques are applied to video frames to create a diverse dataset:

1. Random Horizontal Flip:
   First, we randomly choose if the video should be flipped or not, if yes, all frames are flipped horizontally. This augmentation introduces variability in the orientation of the objects within the frame, making the model robust to mirror images.

2. Random Rotation:
   All frames are rotated by a random angle within a specified range (-30, 30) degrees. This augmentation helps the model handle different orientations of the objects.

3. Color Jitter:
   The brightness, contrast, saturation, and hue of all frames of one video are randomly adjusted. This technique simulates different lighting conditions

and color variations in the input data. Random factors for brightness, contrast, saturation, and hue are generated and applied consistently across all frames in a video.

By applying these augmentation techniques, the video dataset becomes more diverse, helping the model learn to handle various real-world scenarios and reducing the risk of overfitting. This approach ensures that the trained model performs well on unseen data, making it robust and reliable.

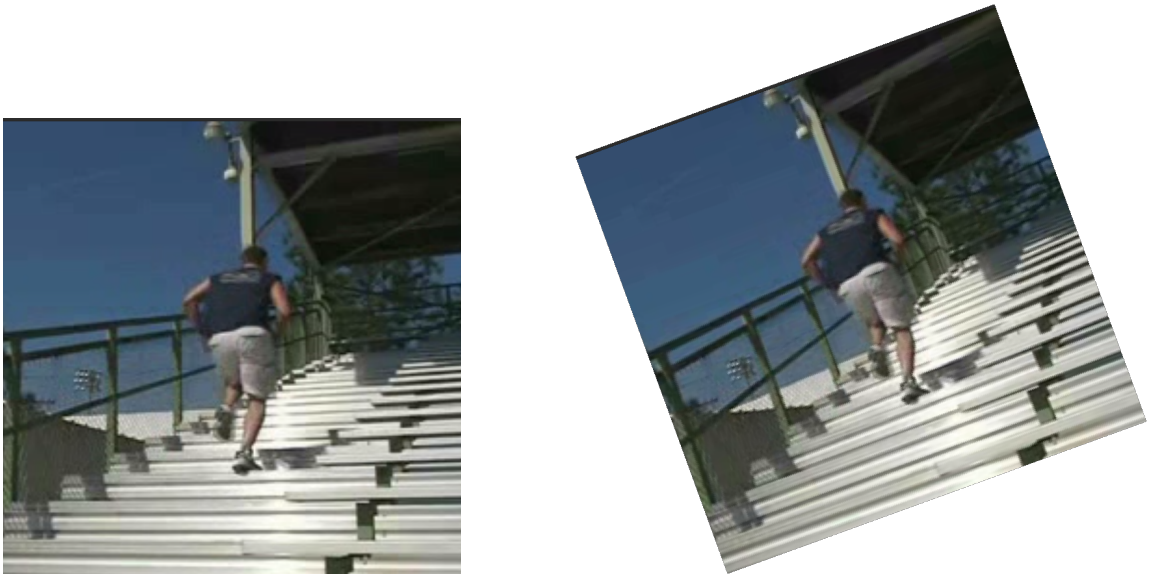

Figura 2.4: Augmentation- rotate of a frame for 20 degrees

2.1.7 t-SNE and K-means Clustering Analysis

To better understand the feature space obtained from the DINOv2 model, we performed a t-SNE (t-Distributed Stochastic Neighbor Embedding) and k-means clustering analysis. This analysis helps visualize the high-dimensional feature space in a two-dimensional plane and identify potential clusters within the data.

We applied t-SNE to reduce the dimensionality of the DINOv2 features from 1536 dimensions to 2 dimensions, allowing for easier visualization and interpretation of the feature distributions. The t-SNE visualization revealed distinct groupings of features, indicating that the DINOv2 model effectively captures meaningful patterns in the video data.

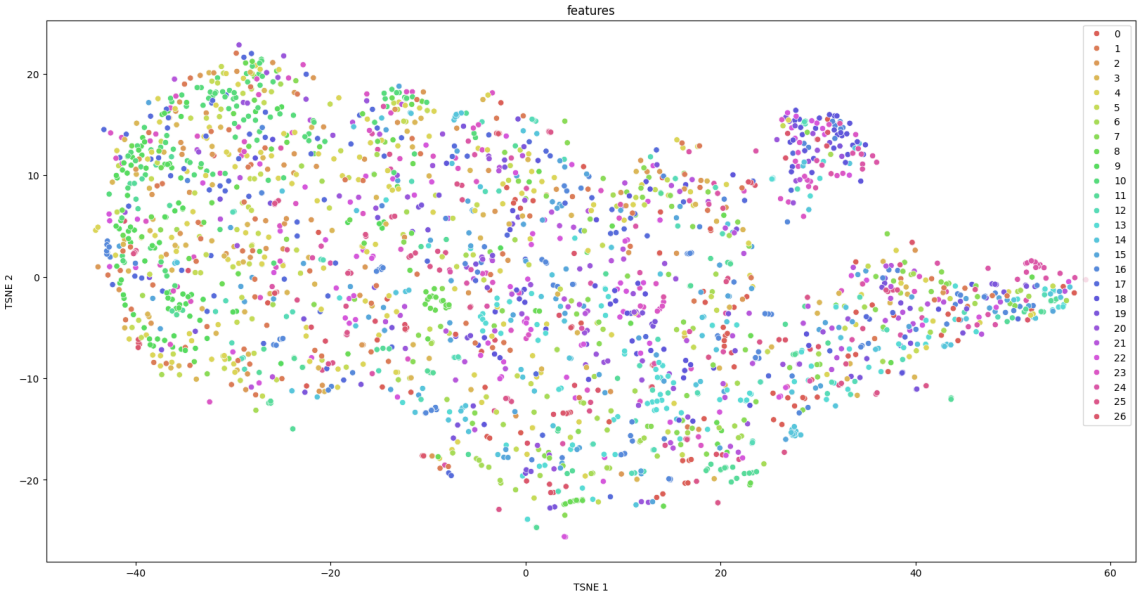

Figura 2.5: t-SNE visualization of the feature space extracted by the DINOv2 model. Each point represents a feature vector, colored by action class. The plot highlights the model's ability to capture distinct patterns corresponding to different actions, with visible clusters indicating high intra-class similarity and regions of overlap suggesting areas where actions share similar features.

The t-SNE plot provided above offers a visual representation of the high-dimensional feature space obtained from the DINOv2 model, reduced to two dimensions. Each point in the plot represents a feature vector from the dataset, with different colors indicating distinct action classes.

Key Observations in the figure 2.5
Distinct Clusters: The plot reveals several distinct clusters, indicating that the DINOv2 model effectively captures meaningful patterns in the video data. These clusters suggest that the model can differentiate between various action classes.
Class Separation: Some classes are more tightly clustered, indicating higher intra-class similarity, while others are more spread out, suggesting greater variability within those classes. For example, a dense cluster of points with the color representing "running" (number 4 on the figure) likely indicates many similar samples of running and activities that form clear, distinct clusters, such as "sky diving" (number 20) or "rock climbing" (number 17), which are likely to have unique, distinguishable features.
Overlap: There are regions where points of different colors overlap, indicating that certain actions share similar feature representations. This could be due to the similarity in the physical movements involved in those actions or limitations in the feature extraction process. Activities that are spread out or overlap with other activities might share similar features. For instance, if "walking" (number 1) and "jogging" (number 10) are not well-separated, they might share common attributes that t-SNE finds challenging to differentiate.
Density: The density of points within clusters varies, which might reflect the number of samples available for each class or the complexity of the action being represented.

Additionally, we used k-means clustering to classify the features into k-distinct clusters. This unsupervised learning technique allowed us to explore the natural groupings of the features without prior knowledge of the class labels. The clustering results provided further insights into the similarity and dissimilarity among the features extracted from different video frames.

These feature vectors were then subjected to K-Means clustering with the following parameters:

Number of Clusters: 1000, chosen to capture a wide range of caloric values.
Random State: 123, to ensure reproducibility of the results.

Number of Initializations: 30, to ensure robust clustering by running the algorithm with different initial centroids and selecting the best output.
Maximum Iterations: 2000, to ensure convergence of the algorithm.

Each feature vector was assigned to one of the 1000 clusters, each cluster representing a specific caloric value. The caloric value for each cluster was determined by the centroid of the cluster. The estimated caloric value for each feature vector (video) was compared to the ground truth caloric value. The mean absolute error (MAE) was calculated. The resulting MAE for the K-Means clustering approach was found to be 303.42, indicating the average deviation of the estimated caloric values from the actual caloric values.

2.1.8 Estimating Hourly Energy Cost from Video Input

Given a video input, the objective of this study is to estimate the hourly energy expenditure associated with the activity performed by the individual depicted in the video. Importantly, the focus is on the intensity of the bodily activity rather than its duration, aiming to infer the number of kilocalories burnt per hour.

Due to the continuous nature of the caloric values, regression-based losses, such as the Euclidean L2 loss, would typically be appropriate for this task. However, it was observed that regression optimization tends to converge to a constant value in this context, a phenomenon also noted in other multimodal problems. To address this issue, the problem is reformulated as a multinomial classification task with additional label softening.

To achieve this, each caloric value annotation, l, is binarized with a resolution of 1 kcal within a specified range $n \in [0, N]$, where N is set to 1000 kcal. To retain certain regression properties, such as penalizing predictions that fall closer to the ground-truth bin less severely, the labels are softened using a Gaussian distribution with a given standard deviation σ (in this project, it is set to 5). For each ground truth annotation l, the softened label is represented as a distribution over N bins:

$$l_s[n] = \frac{1}{\sigma\sqrt{2\pi}} exp(-\frac{(n-l)^2}{2\sigma^2})$$

where l_s denotes the soft label used for supervision. This method is directly taken from [14]. Losses like Kullback-Leibler (KL) divergence, which measures

the distance between two distributions, are used between the ground truth and predicted distributions, to guide the training process.

By converting the problem into a classification task with softened labels, this approach leverages the strengths of both regression and classification methods, enhancing the model's ability to accurately estimate hourly energy expenditure from video data.

2.2 Model Architectures

The task of estimating hourly energy expenditure from video inputs involves a multi-step process utilizing advanced model architectures. In this study, foundation models DINOv2 and CLIP are employed for feature extraction, followed by specialized networks for calorie estimation.

2.2.1 Foundation Models for Feature Extraction

DINOv2 Model

As it is explained in [31], the DINOv2 architecture builds on the success of self-supervised learning methods, combining elements from DINO, iBOT, and SwAV to effectively learn visual representations without requiring labeled data. The model utilizes a Vision Transformer (ViT) framework, which processes video frames by dividing each frame into patches and embedding these patches as input tokens. Key components of the DINOv2 architecture include image-level and patch-level objectives, where cross-entropy losses are computed between the features extracted from a student and a teacher network. In the patch-level objectives, the student network uses random masked tokens. At the same time, the teacher processes visible tokens, and the loss functions are designed to ensure the student learns robust features. Additionally, the architecture incorporates the Sinkhorn-Knopp centering for better normalization, and the KoLeo regularizer to ensure a uniform distribution of features within a batch. To handle high-resolution training efficiently, DINOv2 employs techniques like sequence packing and efficient stochastic depth, which optimize memory usage and computation speed. These advancements make DINOv2 a powerful tool for capturing the spatiotemporal dynamics in videos, crucial for tasks like estimating hourly energy expenditure from video data.

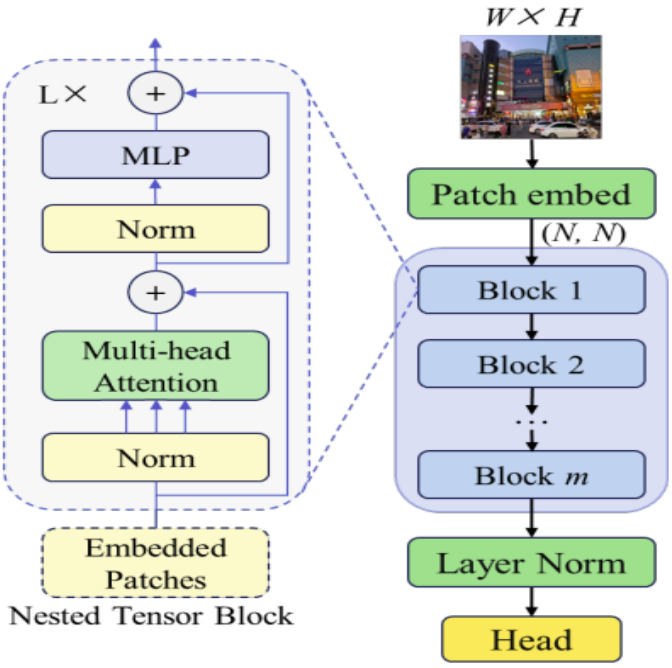

Figura 2.6: The structural diagram of DINOv2 model. Image reference: [41]

In this study, we use DINOv pre-trained model to extract features of videos. we use the torch.hub.load function from the PyTorch library to load a pre-trained model from this repository: facebookresearch/dinov2

CLIP model

Based on [28], CLIP (Contrastive Language-Image Pre-training) is an advanced model architecture designed to learn visual concepts from natural language supervision. It employs a dual-encoder structure comprising an image encoder and a text encoder. The image encoder, either a CNN (ResNet-50) or a Vision Transformer (ViT), processes images to generate visual feature embeddings. Based on a transformer model, the text encoder converts textual descriptions into corresponding embeddings. The core of CLIP's training objective is to align these image and text embeddings using a contrastive learning approach, where the model learns to associate correct image-text pairs and distinguish mismatched pairs through symmetric cross-entropy loss over cosine similarities. This enables CLIP to perform zero-shot learning, where it can generalize to new visual classification tasks without additional task-specific training data. By leveraging a large and diverse dataset of 400 million image-text pairs from the internet, CLIP learns robust, transferable visual representations, significantly enhancing its generalization capabilities across various tasks. To extract features from videos, we utilized

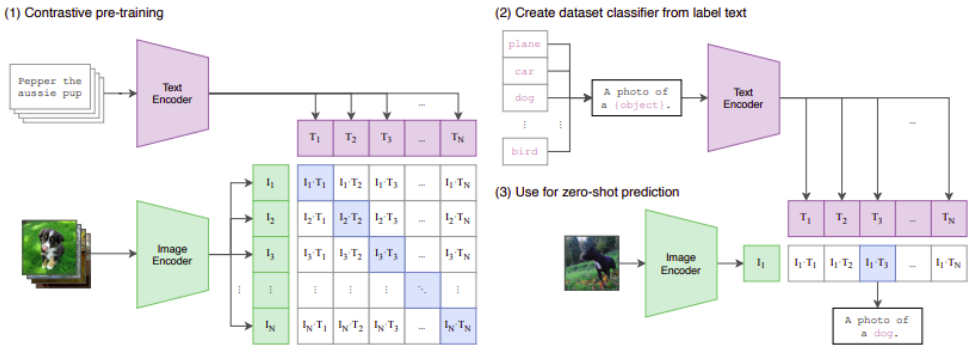

Figura 2.7: Diagram of the CLIP architecture showing the process of contrastive pre-training (1), creating dataset classifiers from label text (2), and using the model for zero-shot prediction (3). Image reference: [28]

the CLIP model by leveraging an open-source implementation available on Gi-

tHub. The repository [42] provides the necessary tools and scripts to process video data and extract meaningful features using the CLIP architecture. This implementation facilitates the integration of natural language supervision into the feature extraction process, enabling robust and generalized representations of visual data.

2.2.2 Calorie Estimation Network

After extracting features using DINOv2 and CLIP, a neural network is used to estimate the calorie expenditure. The network architecture is designed to handle the extracted features and predict kilocalories burnt per hour.

In this architecture, we use MLP blocks. A Multi-Layer Perceptron (MLP) is a class of feedforward artificial neural networks (ANN) that consists of at least three layers of nodes: an input layer, one or more hidden layers, and an output layer. Each node, or neuron, in one layer connects to every node in the next layer, making MLPs fully connected networks. The primary advantage of MLPs lies in their ability to model complex non-linear relationships through their use of activation functions, such as ReLU (Rectified Linear Unit). This capability allows MLPs to learn and approximate complex functions, making them highly effective for a variety of tasks, including classification, regression, and feature extraction. MLPs have been widely used due to their flexibility and effectiveness in handling diverse types of data and problems. They are especially useful in scenarios where the relationship between input and output is not straightforward, enabling the network to learn hierarchical representations of data. The effectiveness of MLPs is well-documented in various studies, highlighting their role in advancing machine learning applications across different domains[43].

The architecture for calorie estimation consists of a Multi-Layer Perceptron (MLP) is the main part of an Evaluator framework to estimate specific outcomes such as calorie expenditure and activity classification. The MLP features a series of fully connected layers: the input layer processes the initial feature vectors, followed by three hidden layers with sizes 1536 (feature size) to 256, 256 to 128, and 128 to the output dimension, respectively. Each hidden layer is activated using the ReLU function to introduce non-linearity, while the final layer's output is passed through a softmax function to generate probabilistic predictions. In the Evaluator framework, each MLP block processes the feature vectors independently to produce the final output. This architecture enables robust and flexible mo-

deling, capturing complex relationships within the data and facilitating accurate predictions in diverse applications.

2.3 Loss Functions

Our study addresses two key tasks: action classification and calorie estimation, each with distinct loss functions to optimize performance. For the action classification task, which involves 27 classes, we use cross-entropy loss to train the model to accurately classify the actions depicted in the videos. This classification task also serves an auxiliary role in supporting the calorie estimation task. While we do not aim to estimate caloric expenditure based solely on action classes, knowing the action class can significantly aid in more accurately determining the calories burned, as different actions typically have varying caloric costs.

For the calorie estimation task, we convert continuous caloric values into discrete bins ranging from 0 to 1000 as explained in 2.1.8, resulting in 1000 output classes from our network. We experiment with different loss functions to train this part of the model: Kullback-Leibler (KL) divergence, Jensen-Shannon divergence, and cross-entropy loss. These losses help the model learn a probability distribution over the caloric bins, to estimate the caloric expenditure from the video data accurately.

The primary training strategy involves using the sum of the losses from both the action classification and calorie estimation tasks. This combined loss approach ensures that the model learns to perform both tasks effectively.

2.3.1 Kullback-Leibler loss function

Kullback-Leibler (KL) divergence is a measure from information theory that quantifies the difference between two probability distributions P and Q. Specifically, it measures how much information is lost when Q is used to approximate P. Mathematically, it is defined as:

$$D_{kl}(P||Q) = -\sum_{x \in X} P(x).log\frac{Q(x)}{P(x)}$$

In our context, P represents the ground truth calorie distribution, and Q represents the predicted calorie distribution.

below are some of the advantages of this loss function:
Sensitivity to Differences: KL divergence is very sensitive to differences between the predicted and actual distributions, which helps the model learn to make precise adjustments to predictions. This is particularly useful in applications like speech source separation, as highlighted by [44].
Information Theoretic Basis: It provides a solid theoretical foundation for measuring how one distribution diverges from another, making it a preferred choice in many machine-learning applications.
Handling Probabilistic Outputs: KL divergence is particularly effective for tasks involving probabilistic outputs, as it directly compares the full distributions rather than just point estimates.

Some disadvantages could be effective in the training process:
Sensitivity to Zeroes: KL divergence can be problematic when the predicted distribution contains zero probabilities, leading to undefined values. This requires careful handling and smoothing of the distributions to avoid mathematical issues.
Asymmetry: KL divergence is asymmetric, meaning $D_{kl}(P||Q) \neq D_{kl}(Q||P)$. This can lead to biased learning if not properly managed, as discussed in [45].
Gradient Instability: The high sensitivity to differences can also lead to unstable gradients during training, requiring careful tuning of the learning rate and other hyperparameters.

2.3.2 Jensen-Shannon loss function

Jensen-Shannon (JS) divergence is a symmetrized and smoothed version of Kullback-Leibler (KL) divergence. It measures the similarity between two probability distributions by calculating the average KL divergence of each distribution to the average distribution. Mathematically, JS divergence is defined as:

$$D_{js}(P, Q) = \frac{1}{2} D_{kl}(P||M) + \frac{1}{2} D_{kl}(Q, M)$$

where $M = \frac{1}{2}(P + Q)$ is the average distribution of P and Q. This divergence provides a measure that is always finite and bounded, making it a practical choice for many applications.

Some of the advantages are mentioned here:
Symmetry: Unlike KL divergence, JS divergence treats both distributions equally,

which helps in avoiding the bias introduced by the asymmetry of KL divergence. This ensures a balanced measure of divergence.
Stability: JS divergence is more stable than KL divergence, particularly when dealing with distributions that have zero probabilities. The smoothing effect of averaging the distributions mitigates issues related to undefined values and instability.
Handling Noisy Labels: The JS divergence is effective in scenarios with noisy labels, as it can provide a more robust measure compared to KL divergence, making it useful in applications like learning with noisy labels as it is mentioned in [46].

Some disadvantages are listed below:
Complexity: JS divergence is computationally more complex than KL divergence due to the additional step of calculating the average distribution. This can increase the computational cost and complexity of the implementation.
Less Sensitivity: While stability is an advantage, it can also be a disadvantage because JS divergence might be less sensitive to small differences between distributions compared to KL divergence. This reduced sensitivity can sometimes result in less precise adjustments during training.

2.3.3 Cross-Entropy Loss

Cross-entropy loss, also known as log loss, measures the performance of a classification model whose output is a probability value between 0 and 1. It quantifies the difference between two probability distributions: the true distribution (ground truth) and the predicted distribution. For a single instance, cross-entropy is defined as:

$$D_{crossent}(P||Q) = -\sum_{x \in X} P(x).logQ(x)$$

where P is the true distribution, Q is the Predicted distribution, and x represents the possible outcomes [47].

some of the advantages of this loss function are mentioned here:
Direct Interpretation: Cross-entropy loss directly measures how well the predicted probabilities match the true labels, making it straightforward to interpret and optimize. This is particularly useful for classification tasks where the goal is to maximize the likelihood of the correct class.
Effective for Classification: It is highly effective for tasks involving multiple

classes, as it penalizes incorrect predictions by considering the entire probability distribution over all classes.
Gradient-Friendly: Cross-entropy loss provides smooth and well-behaved gradients, which are essential for the efficient training of deep neural networks. This makes it a popular choice for training models using gradient-based optimization techniques.

It is needed to point out some of the disadvantages:
Non-Continuous Targets: Cross-entropy loss assumes discrete classes, which can be limiting when dealing with continuous outputs, such as regression tasks. In such cases, other loss functions like mean squared error (MSE) might be more appropriate.
Gradient Vanishing/Exploding: Although generally gradient-friendly, cross-entropy loss can still suffer from vanishing or exploding gradients in very deep networks, particularly when the predictions are extremely confident but incorrect. This requires careful tuning of learning rates and network initialization.
Sensetive to Noise: Cross-entropy loss can be sensitive to noisy labels, as incorrect labels can disproportionately affect the loss, leading to poor generalization. This necessitates techniques like label smoothing or robust loss functions to mitigate the impact of noise.

2.4 Experimental Setup

In our experimental setup, we aim to evaluate the performance of our model architecture in both classifying video actions and estimating caloric expenditure. We utilize two foundation models, DINOv2 and CLIP, for feature extraction from video frames. The dataset is divided into training, test, and cross-test sets. The training set is used to optimize model parameters, the test set for evaluating the performance, and the cross-test is used to study the ability of the model to generalize to unseen categories.

The feature extraction process begins with preprocessing the video data, where each video is divided into frames and resized to 224x224 pixels. For DINOv2, we apply a sequence of transformations including center cropping, and normalization before feeding the frames into the model. CLIP, on the other hand, processes the entire video directly to extract features.
These extracted features are then input into our Multi-Layer Perceptron (MLP)

network, which consists of multiple fully connected layers activated by ReLU functions and a final softmax layer for classification.

Our evaluation metrics include the accuracy of action classification and the precision of calorie expenditure predictions. The action classification task involves categorizing each video into one of 27 predefined classes using cross-entropy loss. The caloric estimation task treats the caloric values as continuous but binarizes them into 1000 discrete bins, applying various losses to enhance training effectiveness.

Additionally, data augmentation techniques such as Gaussian noise addition, feature dropout, and rotating the video frames are employed to mitigate the overfitting problem when it occurs. Regularization techniques such as L2 weight decay are also applied to prevent overfitting and enhance the model's robustness.

By carefully selecting and tuning these parameters and loss functions, we aim to build a robust and accurate model capable of both classifying video actions and estimating caloric expenditure. The integration of action classification as a supporting task ensures that the model leverages the contextual information from the action classes to enhance the precision of calorie estimation.

Capitolo 3

Results

3.1 Evaluation Metrics

To evaluate the performance of our model, we adopt several metrics that provide a comprehensive assessment of both the accuracy and the quality of the predictions. The metrics are similar to metrics used in [14].

1. **Mean Absolute Error (MAE)**: This is our primary evaluation metric. MAE measures the average magnitude of errors between the predicted values and the ground truth, without considering their direction. It is defined as:

$$\text{MAE} = \frac{1}{n}\sum_{i=1}^{n}|y_i - \hat{y}_i|$$

   where y_i represents the actual kilocalorie values for a video and $\hat{y}_i$ represents the predicted kilocalorie values. To calculate it, we consider the *argmax* of the output of the network. MAE is an intuitive and straightforward metric that reports the mean discrepancy between the predictions and the ground truth in the target units, which in this case are kilocalories. A lower MAE indicates better predictive accuracy.

2. **Spearman Rank Correlation (SPC)**: SPC evaluates the strength and direction of the monotonic relationship between the predicted and actual values. It is defined as:

$$\rho = 1 - \frac{6\sum d_i^2}{n(n^2-1)}$$

where d_i is the difference between the ranks of the actual and predicted values, and n is the number of observations. SPC ranges from -1 to +1, where +1 indicates a perfect positive monotonic relationship, -1 indicates a perfect negative monotonic relationship, and 0 indicates no monotonic relationship. While SPC is useful for understanding the association between the predictions and the ground truth, it should be interpreted with caution as it does not account for the scaling and shifting of the data. For example, if the predictions consistently overestimate or underestimate the actual values by the same amount, SPC will still show a high correlation. We use the scipy.stats.spearmanr function from the SciPy library to calculate SPC.

3. **Negative Log-Likelihood (NLL)**: NLL measures the quality of probabilistic predictions by quantifying the likelihood of the ground truth labels given the predicted probabilities. NLL is computed as:

$$NLL = -(y_{true} log(y_{pred}) + (1 - y_{true}) log(1 - y_{true}))$$

where y_{true} is the Gaussian distribution made by the true calorie assigned to the sample and the y_{pred} is the predicted probability. Lower NLL values indicate that the model assigns higher probabilities to the observed outcomes, thus reflecting better model performance. NLL is particularly important in contexts where the uncertainty of predictions needs to be quantified and minimized.

By employing these metrics, we ensure a robust evaluation of our model's performance. MAE provides a direct measure of predictive accuracy, SPC offers insights into the strength of the relationship between predictions and actual values, and NLL evaluates the probabilistic quality of the predictions. Together, these metrics allow us to comprehensively assess the effectiveness of our approach in estimating caloric expenditure from video data.

3.2 Implementation Details

In this section, we detail the setup of our experiments, including the hardware and software environments, the configuration of the models, and other relevant aspects that were critical to the execution of our study.

Hardware and Software Our experiments were conducted using Google Colab Pro with NVIDIA T4 GPUs. The primary software environment consisted of:

- Operating System: Ubuntu 20.04 LTS (within Google Colab environment)
- Programming Language: Python 3.8
- Deep Learning Framework: PyTorch 1.10
- Additional Libraries: NumPy, SciPy, OpenCV, torchvision, and others

The use of NVIDIA T4 GPUs enabled us to handle the computationally intensive tasks of training deep neural networks, particularly with large-scale video data.

Model Configuration We utilized two foundation models, DINOv2 and CLIP, for feature extraction from video frames. The specific configurations for these models were as follows:

- DINOv2: We used the `dinov2_vits14` variant, which was loaded using the torch.hub module from the facebookresearch/dinov2 repository. The model was configured to output intermediate layers for feature extraction.
- CLIP: The CLIP model used was ViT-B/32, which was loaded using the openai/clip repository. The model was configured to handle entire videos without the need for pre-extraction of frames.

Training Configuration Our models were trained using the ADAM optimizer with the following hyperparameters:

- Weight Decay: 1e-5
- Batch Size: 4
- Learning Rate: 1e-4
- Number of Epochs: 200 (and an additional run for 700 epochs)

These hyperparameters were chosen based on preliminary experiments and literature recommendations to ensure effective convergence and generalization of the models.

3.3 Results

In this section, we present the outcomes of our experiments, including detailed analyses of the performance metrics, graphical representations of the results, and comparisons of different training configurations and loss functions.

We conducted a series of experiments using various configurations and loss functions, including Kullback-Leibler Divergence (KLD), Jensen-Shannon Divergence (JSD), and Cross-Entropy (CE). Each model was trained for different epochs and the performance was evaluated using key metrics: Mean Absolute Error (MAE), Spearman Rank Correlation (SPC), and Negative Log-Likelihood (NLL).

3.3.1 feature extraction dimensions

Average over temporal features

The extracted features serve as inputs for our calorie estimation model. The features extracted using the DINOv2 model have a dimension of 160 × 1536. Here, 160 represents the temporal dimension corresponding to the video frames, and 1536 is the feature dimension for each frame. The CLIP model provides features with a dimension of number of frames × 512. For standardization, we select 90 frames randomly, resulting in a feature dimension of 90 × 512.

To make the features compatible with the input requirements of the evaluator for calorie estimation, which expects a feature vector of dimensions 1 × 1536 and 1 × 512, we computed the mean over the temporal dimension for both DINOv2 and CLIP features. This averaging operation reduces the feature dimensions to 1 × 1536 and 1 × 512, respectively.

Aggregation

Another way to reduce the frame number dimension to 1, is to design an aggregation process using a series of convolutional and pooling layers. This process ensures that the features are optimally prepared for input into the evaluator network. The detailed steps of the aggregation process are as follows:
Convolutional Layers: The features are passed through a series of convolutional layers. Each layer applies a convolution operation followed by a ReLU activation function, which introduces non-linearity and helps in learning complex patterns in the data.

Pooling Layers: After each convolutional layer, a pooling layer is applied to reduce the spatial dimensions of the feature maps. Pooling helps in reducing the computational load and controlling overfitting by summarizing the presence of features in patches of the feature map.
Fully Connected Layer: The final pooled feature maps are then passed through a fully connected (fc) layer, which consolidates the features into a compact representation suitable for the evaluator.

This aggregation process effectively reduces the dimensionality of the feature vectors while preserving the essential information needed for accurate calorie estimation. The resultant feature vectors are then fed into the evaluator network, which predicts the caloric expenditure based on the processed features.

In summary, the combination of feature extraction using DINOv2 and CLIP, followed by the designed aggregation process, ensures that the features are appropriately scaled and transformed, making them suitable for accurate calorie estimation. This structured approach not only enhances the efficiency of the model but also improves its performance in estimating the calories burned during different activities.

3.3.2 CLIP

1. **Kullback-Leibler Divergence (KLD) Loss** Here there are some results for training the network with KLD loss function for 200 epochs.

   As we observe overfit in figure 3.1, we decide to tune the learning rate and weigh decay to see the changes.

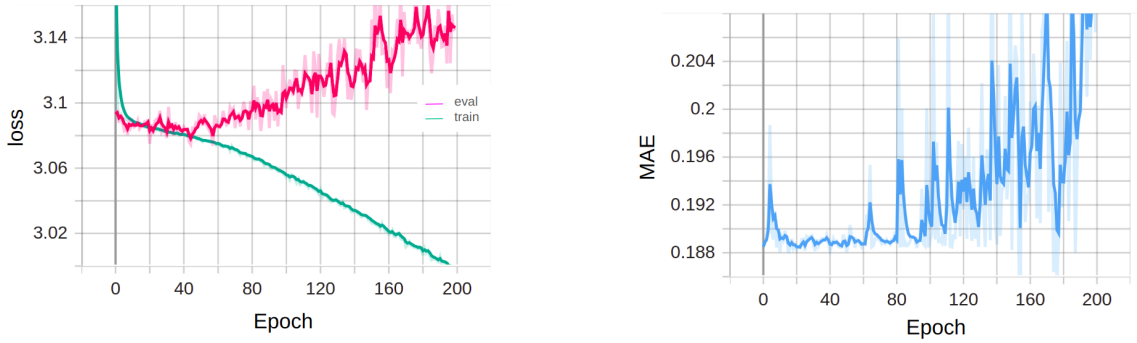

Figura 3.1: Training and validation loss trends for KLD over 200 epochs without learning rate adjustment. The right plot shows the Mean Absolute Error (MAE) trend during training.

change in learning rate and weight decay

- The learning rate is initially set to 1e-4 and here is reduced by a factor of 0.5 every 20 steps.

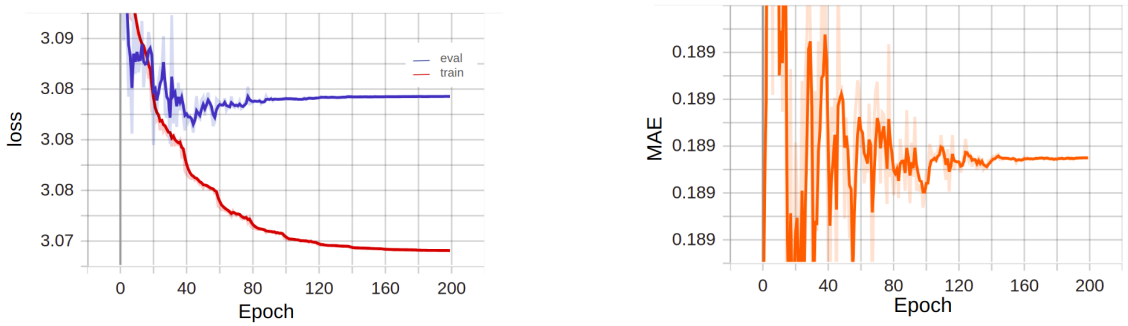

Figura 3.2: Training and validation loss trends for KLD over 200 epochs with learning rate and weight decay adjustment. The right plot shows the MAE trend during training.

- weight decay was set to 1e-4 and The learning rate was initially set to 1e-4 and was reduced by a factor of 0.5 every 20 steps.

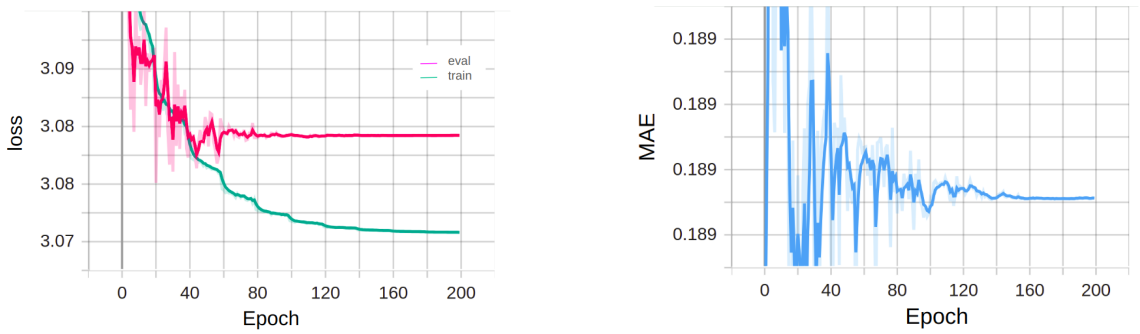

Figura 3.3: Training and validation loss trends for KLD over 200 epochs with learning rate and weight decay adjustment. The right plot shows the MAE trend during training. The right plot shows the MAE trend during training.

Most of the tried configurations for weight decay and learning rate could solve the overfit in the training process but it stops models from learning enough.

2. **Jensen-Shannon Divergence (JSD) Loss** The next loss function we tried was JSD; the result is below. The plot 3.4 shows the overfit after 40 epochs.

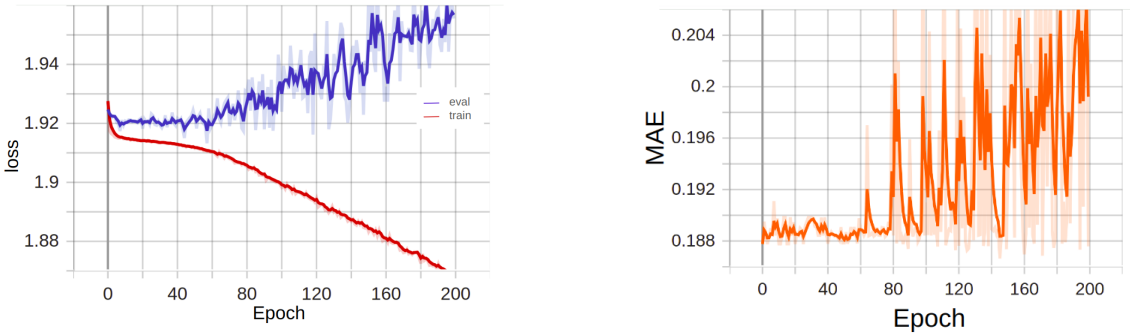

Figura 3.4: Training and validation loss trends for JSD over 200 epochs. The right plot shows the Mean Absolute Error (MAE) trend during training.

3. **Cross-Entropy (CE) Loss** The Cross-Entropy is the third loss function, which is shown below and the overfit is still observable.

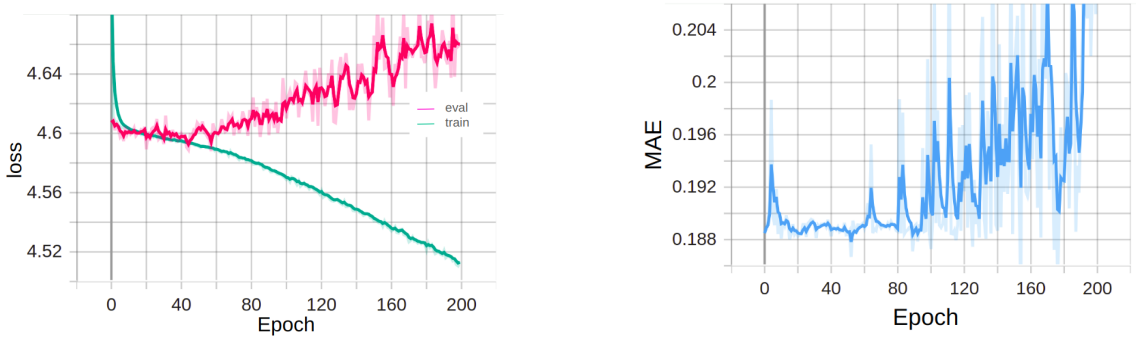

Figura 3.5: Training and validation loss trends for CE over 200 epochs. The right plot shows the Mean Absolute Error (MAE) trend during training.

In our experiments with features extracted from CLIP, we encountered significant overfitting when applying Kullback-Leibler Divergence (KLD), Jensen-Shannon Divergence (JSD), and Cross-Entropy (CE) loss functions. This overfitting persisted despite various adjustments to the learning rate (LR) and weight decay (WD). Initial attempts to tune these hyperparameters were met with limited success; reducing LR or increasing WD to mitigate overfitting resulted in the model's inability to learn effectively, suggesting that the adjustments either failed to address the complexity of the features or overly constrained the learning process. This outcome highlights the challenge of balancing regularization and model capacity in scenarios where high-dimensional and complex features are involved. These findings underline the necessity for more sophisticated regularization techniques or alternative loss functions tailored to handle the intricacies of features derived from advanced models like CLIP.

Augmentation

Feature Augmentation: To enhance our models' robustness and generalization capability, we applied a series of augmentation techniques at the feature level. Given the initial training dataset comprising 2476 video samples, we implemented the following feature augmentation methods to increase the dataset size and variability effectively, methods are explained in 2.1.6.

- Adding Gaussian Noise
- Feature Dropout
- Feature Scaling
- Random Shifts

By incorporating these augmentation methods, we aimed to address the overfitting issue and improve the model's ability to generalize across different data distributions and scenarios. We effectively quintupled the size of our training dataset, increasing it from 2476 to 12380 samples. This substantial increase in the dataset size is expected to improve the model's performance by providing a more diverse and comprehensive set of training examples, thereby enhancing its ability to generalize to new, unseen data.

- **Kullback-Leibler Divergence (KLD) Loss** Here are some results for training the network with KLD loss function for 200 epochs, after feature augmentation with tuning the learning rate and weight decay. in both figures 3.6 and 3.7 we observe that the model could not learn enough.

  - The learning rate is initially set to 1e-4 and is reduced by a factor of 0.5 with patience of 5 steps based on evaluation loss.

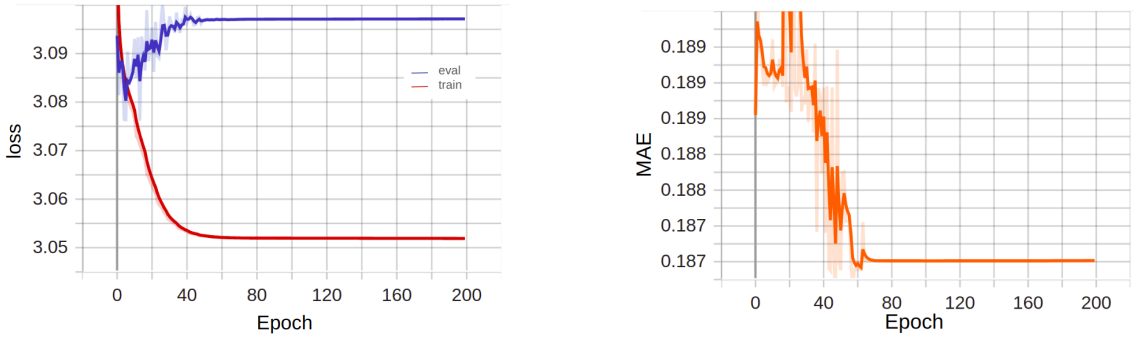

Figura 3.6: The changes of KLD loss function after Feature Augmentation over 200 epochs with learning rate and weight decay adjustment are shown in the left plot, and the right plot shows the trend of MAE during training.

- weight decay is set to 1e-4 and the learning rate is initially set to 1e-4 and is reduced by a factor of 0.5 with patience of 5 steps based on evaluation loss.

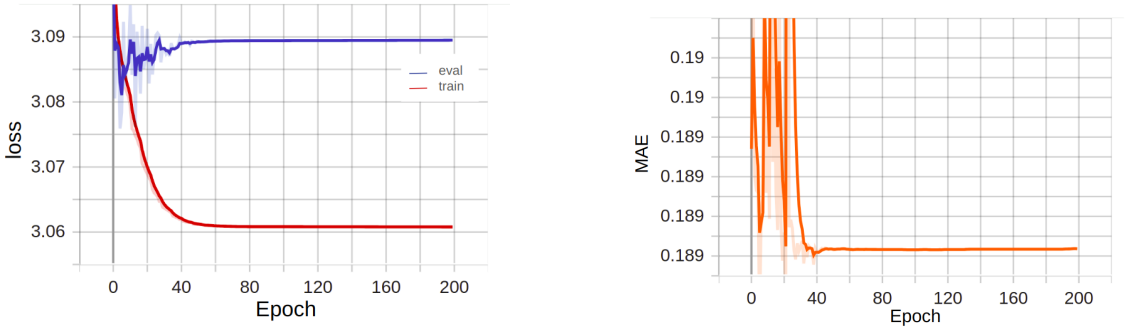

Figura 3.7: The changes of KLD loss function after Feature Augmentation over 200 epochs with learning rate and weight decay adjustment are shown in the left plot, and the right plot shows the trend of MAE during training.

Video Augmentation: We applied a series of augmentation techniques to videos. following video augmentation methods are explained in 2.1.6.

- Random Horizontal Flip
- Random Rotation
- Color Jitter

By applying these augmentation techniques, we quadrupled the size of our training dataset, increasing it from 2476 to 9904 samples.

- **Kullback-Leibler Divergence (KLD) Loss** Here there are some results for the training of the network with KLD loss function for 200 and 700 epochs, after video augmentation.

  - The learning rate was initially set to 1e-4 and was reduced by a factor of 0.5 with patience of 5 steps based on evaluation loss.

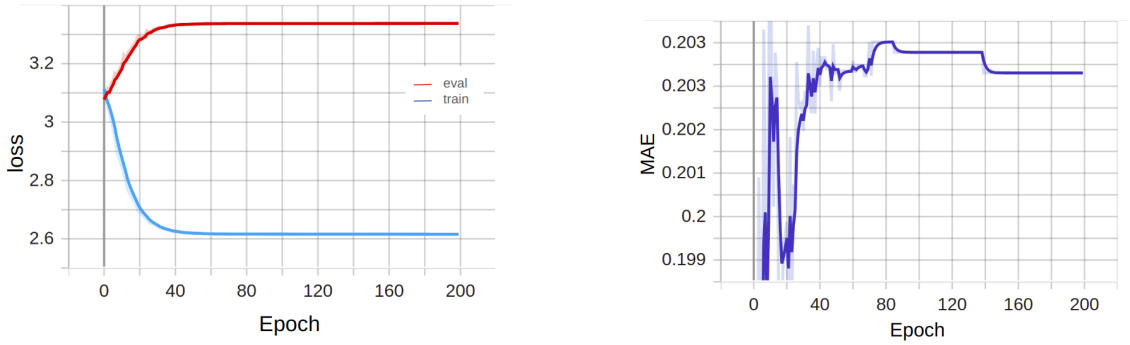

Figura 3.8: The changes of KLD loss function after Video Augmentation over 200 epochs with learning rate and weight decay adjustment are shown in the left plot, and the right plot shows the trend of MAE during training.

- The weight decay was set to 1e-4 and the learning rate was initially set to 1e-4 and was reduced by a factor of 0.5 with patience of 5 steps based on evaluation loss.

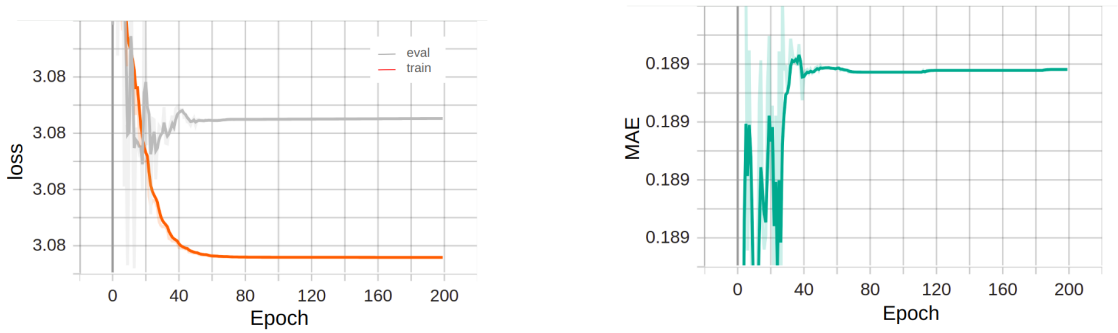

Figura 3.9: The changes of KLD loss function after Video Augmentation over 200 epochs with learning rate and weight decay adjustment are shown in the left plot, and the right plot shows the trend of MAE during training.

- The weight decay was set to 1e-3 and the learning rate was initially set to 1e-4 and was reduced by a factor of 0.5 with patience of 5 steps based on evaluation loss.

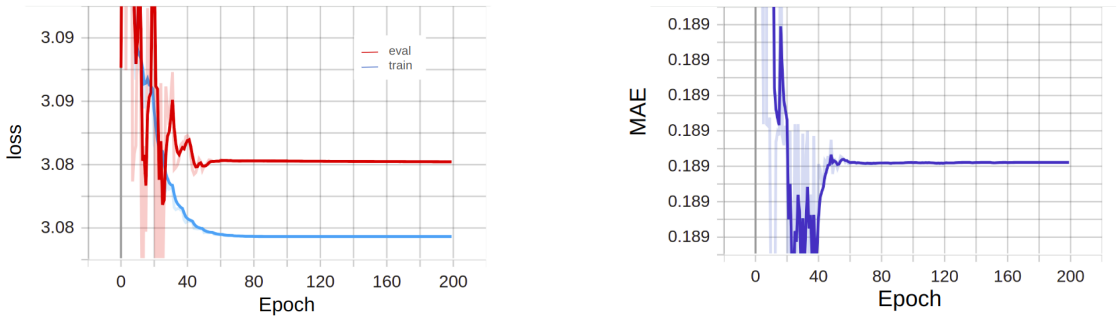

Figura 3.10: The changes of KLD loss function after Video Augmentation over 200 epochs with learning rate and weight decay adjustment are shown in the left plot, and the right plot shows the trend of MAE during training.

- The weight decay was set to 1e-3 and the learning rate was initially set to 1e-4 without decreasing factor.

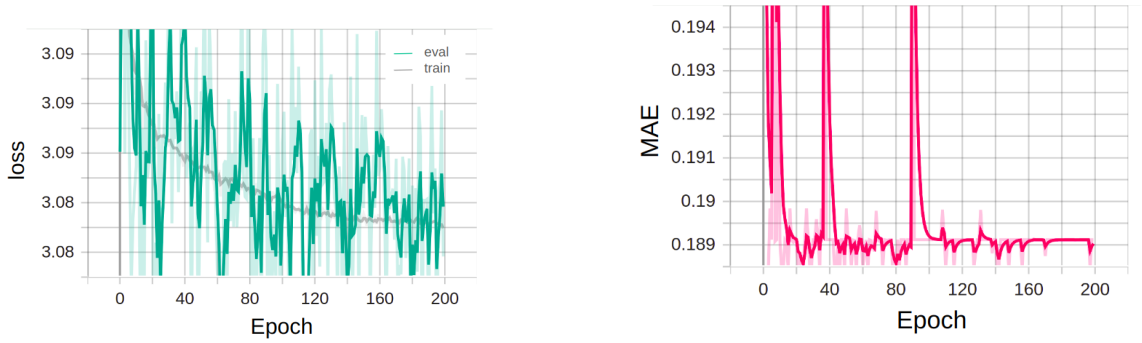

Figura 3.11: The changes of KLD loss function after Video Augmentation over 200 epochs with learning rate and weight decay adjustment are shown in the left plot, and the right plot shows the trend of MAE during training.

To address the persistent overfitting encountered with Kullback-Leibler Divergence (KLD), we applied a combination of feature augmentation, video augmentation, and careful adjustments to the learning rate (LR) and weight decay (WD). Despite these strategies, the results did not improve significantly. The overfitting remained evident or the model's learning capacity was inhibited by the parameter adjustments. Feature augmentation techniques, including the addition of Gaussian noise and feature dropout, were employed to enhance the robustness of the features, while video augmentation aimed to increase the variability of the training data. However, these measures failed to resolve the overfitting issue effectively. Adjustments to LR and WD were similarly unproductive; lowering the LR or increasing WD to combat overfitting either failed to sufficiently regularize the model or excessively constrained it, preventing effective learning.

3.3.3 CLIP + Aggregation

Below are the results of three loss functions, KLD, JSD, and CE with aggregation on the features extracted by clip. In all three training processes, we observe huge overfit.

1. **Kullback-Leibler Divergence (KLD) Loss**

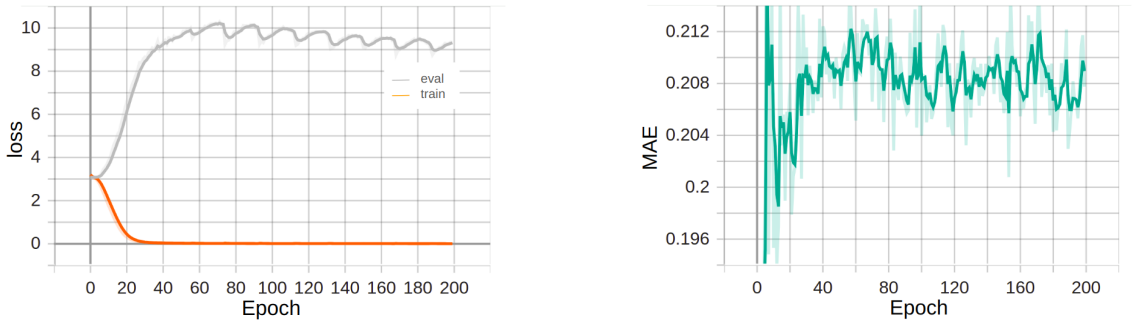

Figura 3.12: Training and validation loss trends for KLD over 200 epochs with aggregation layers applied. The right plot shows the MAE trend during training.

2. **Jensen-Shannon Divergence (JSD) Loss**

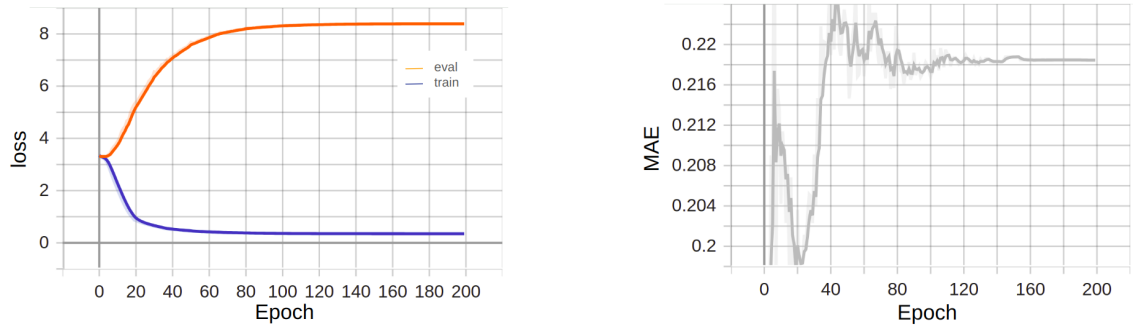

Figura 3.13: Training and validation loss trends for JSD over 200 epochs with aggregation layers applied. The right plot shows the MAE trend during training.

3. **Cross-Entropy (CE) Loss**

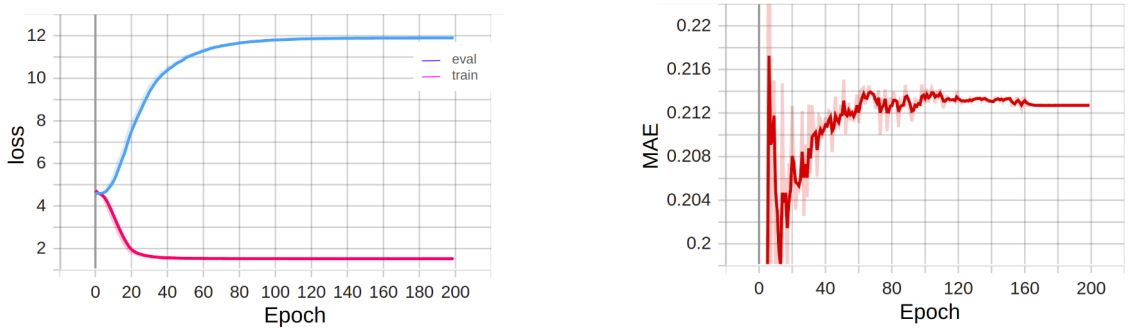

Figura 3.14: Training and validation loss trends for CE over 200 epochs with aggregation layers applied. The right plot shows the MAE trend during training.

In our efforts to enhance the training process, we experimented with replacing getting mean over the temporal dimension with a more complex aggregation technique, hoping it would improve learning across three different loss functions. Despite this adjustment, the issue of overfitting persisted, leading me to reconsider the suitability of CLIP for feature extraction in this context. I hypothesized that CLIP's powerful feature extraction capabilities might be too robust, potentially overshadowing the nuances needed for accurate caloric estimation from video data. This prompted me to explore an alternative approach by experimenting with DINOv2, a different foundation model, to see if it could provide a better balance between feature richness and model generalizability.

3.3.4 DINOv2

Here are some results for training the network with different loss functions, KLD, JSD, and CE for 200 and 700 epochs. For all, after 200 epochs they still can learn therefore we continue to epoch 1000 but after epoch 700, the training process became stable and there were no improvements.

1. **Kullback-Leibler Divergence (KLD) Loss**

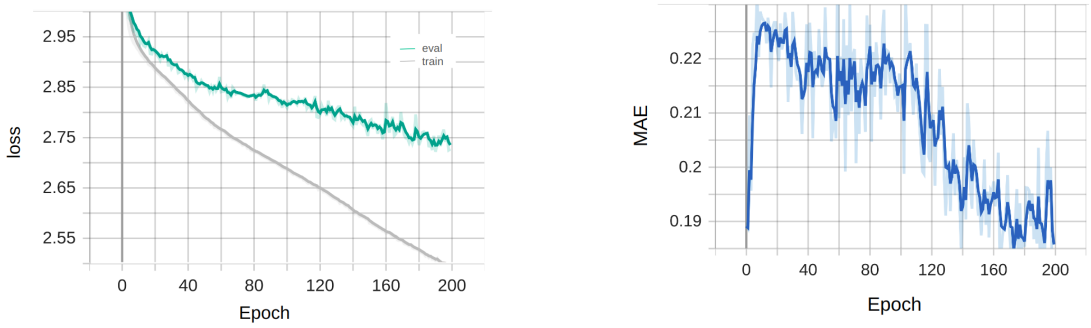

Figura 3.15: Training and validation loss trends for KLD over 200 epochs for DINOv2. The right plot shows the MAE trend during training.

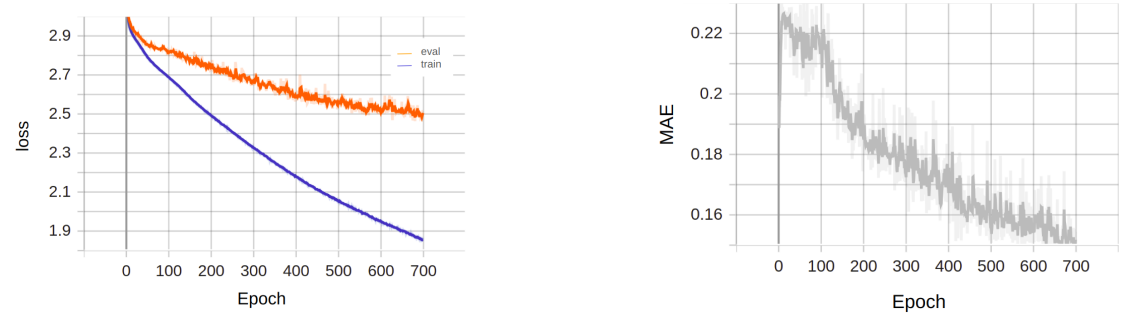

Figura 3.16: Training and validation loss trends for KLD over 700 epochs for DINOv2. The right plot shows the MAE trend during training.

2. **Jensen-Shannon Divergence (JSD) Loss**

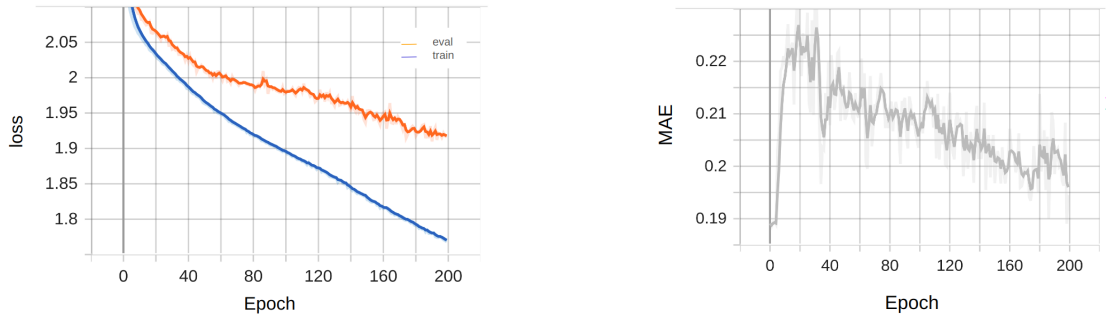

Figura 3.17: Training and validation loss trends for JSD over 200 epochs for DINOv2. The right plot shows the MAE trend during training.

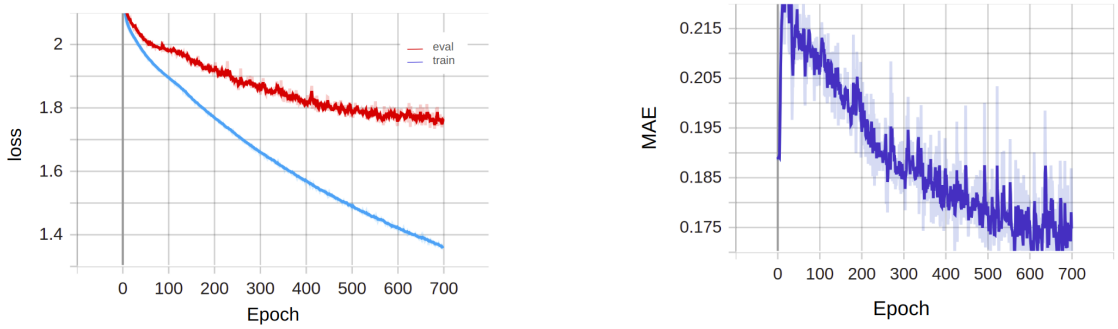

Figura 3.18: Training and validation loss trends for JSD over 700 epochs for DINOv2. The right plot shows the MAE trend during training.

3. **Cross-Entropy (CE) Loss**

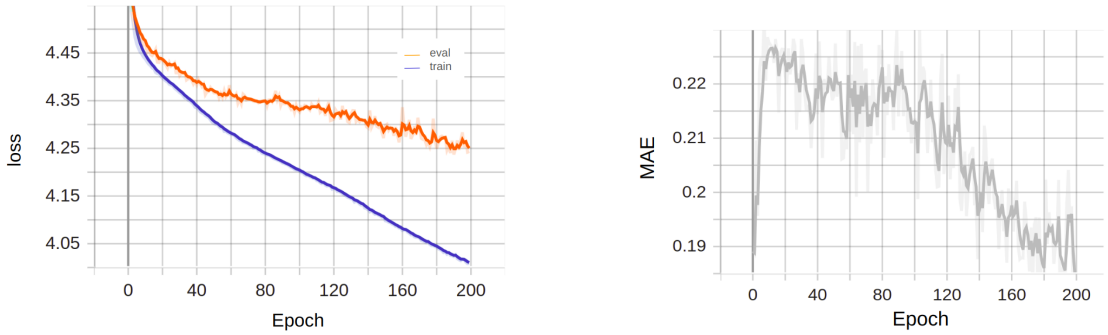

Figura 3.19: Training and validation loss trends for CE over 200 epochs for DINOv2. The right plot shows the MAE trend during training.

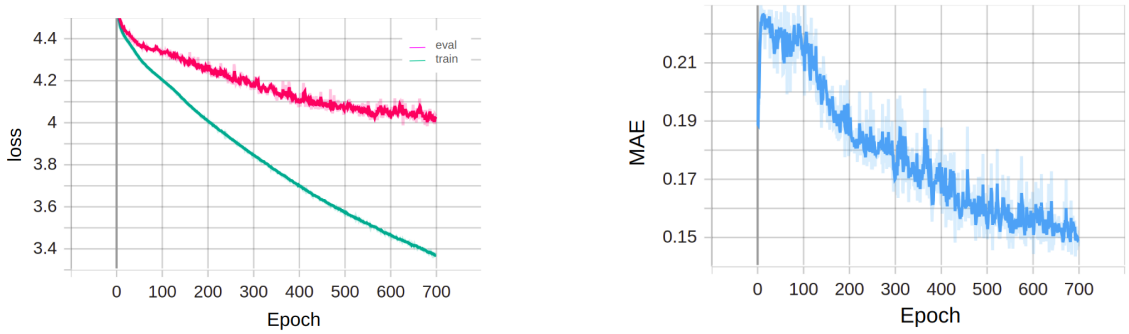

Figura 3.20: Training and validation loss trends for CE over 700 epochs for DINOv2. The right plot shows the MAE trend during training.

Tabella 3.1: comparison between different configurations and loss functions

			known			Unknown		
loss func	epoch	accuracy	MAE	correlation	NLL	MAE	correlation	NLL
KLD	200	22%	181	0.234	6.68	**206**	0.04	10.46
	700	33%	154	0.239	6.52	256	0.04	11.23
JSD	200	21%	194	0.211	8.47	**206**	0.08	12.20
	700	30%	169	0.151	10.31	263	0.12	13.21
CrossEnt	200	23%	179	0.234	6.68	211	0.03	10.45
	700	34%	**148**	0.242	6.53	256	0.04	11.21

In the table 3.1, we compare all the evaluation metrics for known and unknown test sets.

In our experiments utilizing DINOv2 for feature extraction, we evaluated the model's performance across three different losses KLD, JSD, and CE. Each model was initially trained for 200 epochs, during which they all demonstrated commendable performance. To explore the potential for further improvement, we extended the training up to 1,000 epochs. However, it became apparent that after 700 epochs, there were diminishing returns in terms of performance enhancement, leading us to cap the training at 700 epochs. Interestingly, while the Cross-Entropy loss function at 700 epochs yielded the lowest Mean Absolute Error (MAE) of 148 for the known test set, it did not perform as well on the unknown test set, which is critical for assessing the model's ability to generalize. Conversely, the KLD and JSD models trained for only 200 epochs outperformed other configurations on the unknown test set. Notably, the KLD at 200 epochs exhibited superior results in both the known and unknown MAE compared to the JSD, making it the most effective in balancing performance across different test scenarios. This suggests that while longer training durations might refine certain metrics on familiar data, shorter training periods with strategically chosen loss functions like KLD could better enhance overall model robustness and generalization. Same as CLIP, we decided to observe the performance using aggregation.

3.3.5 DINOv2 + Aggregation

We assumed that with aggregation we could have better training results in comparison with average over temporal features. we can see the result of training for all loss functions for 200 and 700 epochs. In epoch 700 for all loss functions we observed overfit, therefore tried to change the learning rate and weight decay to solve it.

1. **Kullback-Leibler Divergence (KLD) Loss**

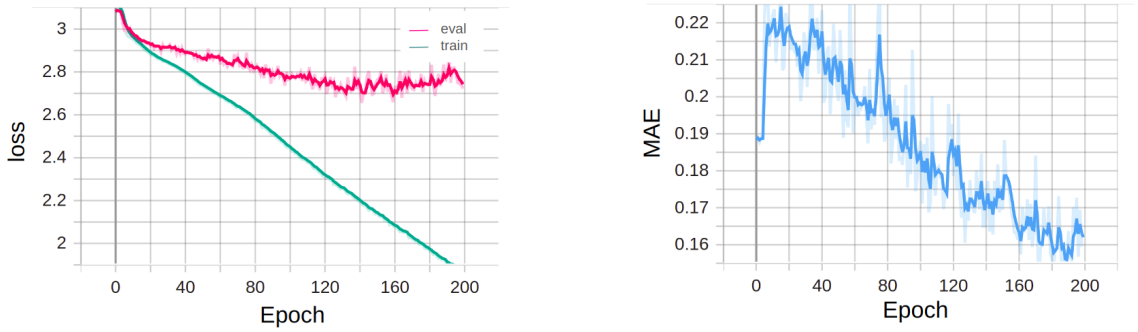

Figura 3.21: The left plot is training and validation loss trends for KLD over 200 epochs with aggregation layers for DINOv2. The right plot shows the MAE trend during training.

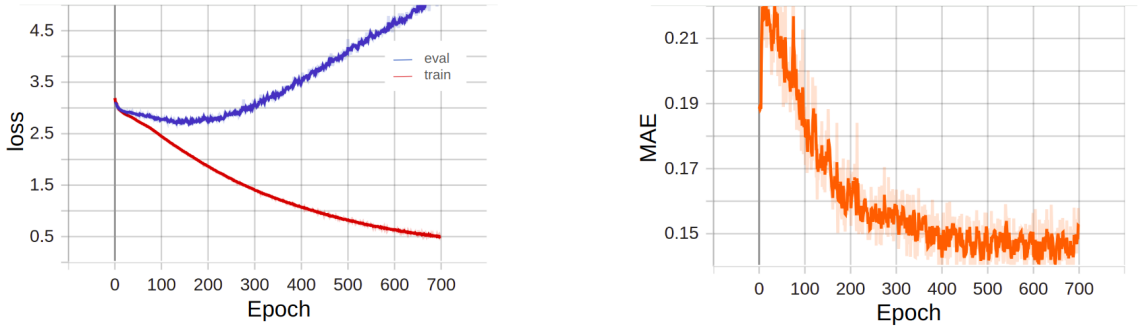

Figura 3.22: The left plot is training and validation loss trends for KLD over 700 epochs with aggregation layers for DINOv2. The right plot shows the MAE trend during training.

change in learning rate and weight decay

- weight decay is set to 1e-4. We still observe the overfit for this configuration.

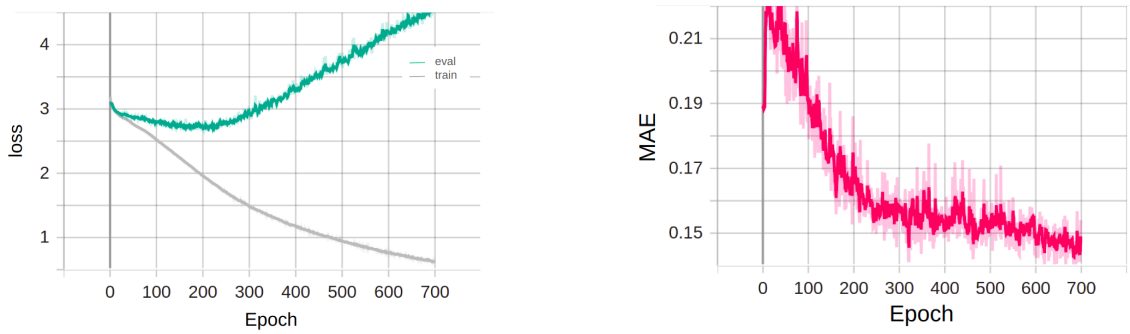

Figura 3.23: The left plot is training and validation loss trends for KLD over 700 epochs with aggregation layers and with learning rate and weight decay adjustment for DINOv2. The right plot shows the MAE trend during training.

- weight decay is set to 1e-5 and the learning rate is initially set to 1e-4 and is reduced by a factor of 0.5 with the patience of 5 steps based on evaluation loss. We also set the stopper to stop the training before the evaluation loss stops decreasing. For figure 3.24 the stopped epoch is 88.

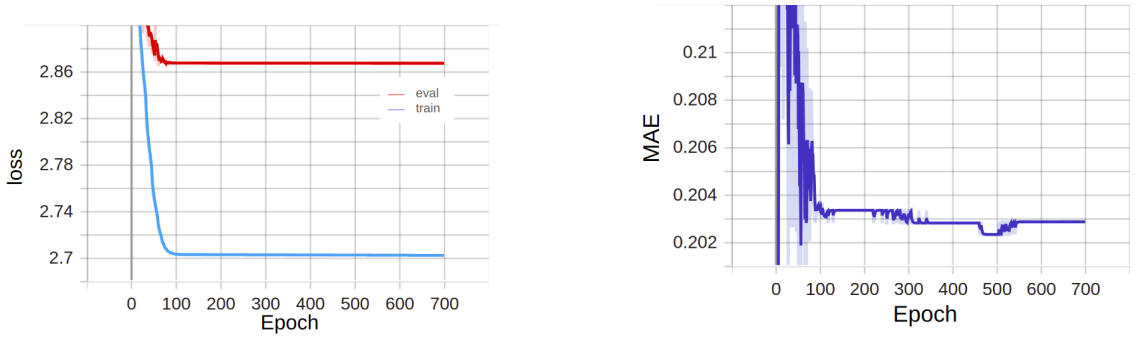

Figura 3.24: The left plot is training and validation loss trends for KLD over 700 epochs with aggregation layers and with learning rate and weight decay adjustment for DINOv2. The right plot shows the MAE trend during training.

- weight decay is set to 1e-5 and the learning rate is initially set to 1e-4 and is reduced by a factor of 0.5 with the patience of 10 steps based on evaluation loss. For figure 3.25 the stopped epoch is 167.

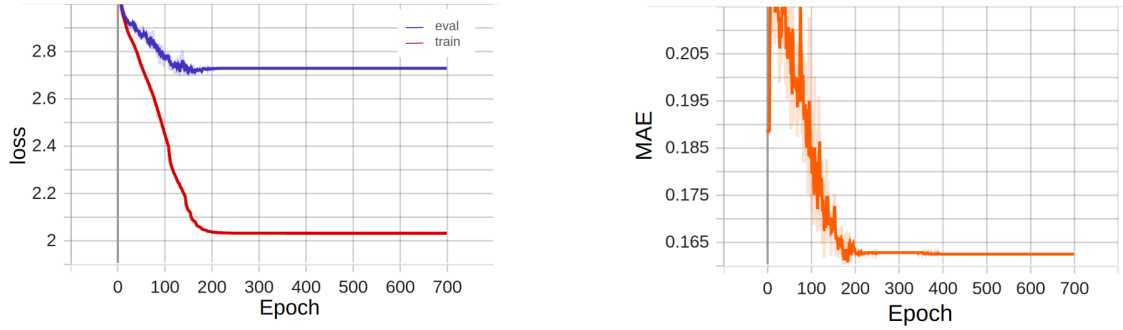

Figura 3.25: The left plot is training and validation loss trends for KLD over 700 epochs with aggregation layers and with learning rate and weight decay adjustment for DINOv2. The right plot shows the MAE trend during training.

- weight decay is set to 1e-5 and the learning rate was initially set to 1e-4 and was reduced by a factor of 0.5 with the patience of 15 steps based on evaluation loss. For figure 3.26 the stopped epoch is 173.

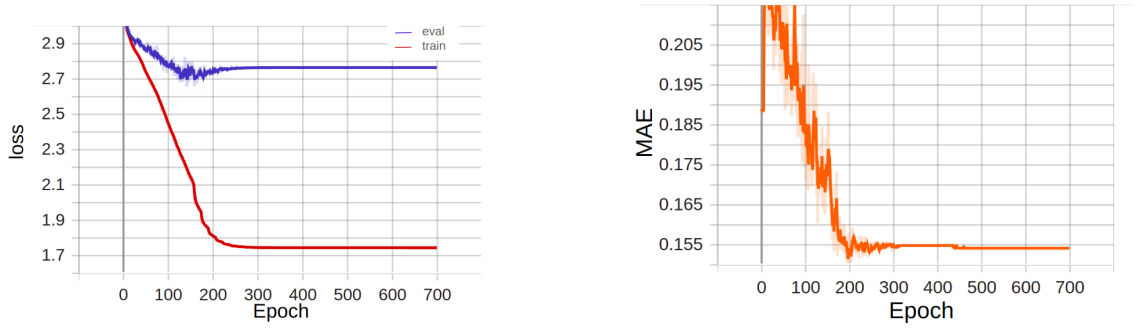

Figura 3.26: The left plot is training and validation loss trends for KLD over 700 epochs with aggregation layers and with learning rate and weight decay adjustment for DINOv2. The right plot shows the MAE trend during training.

2. **Jensen-Shannon Divergence (JSD) Loss**

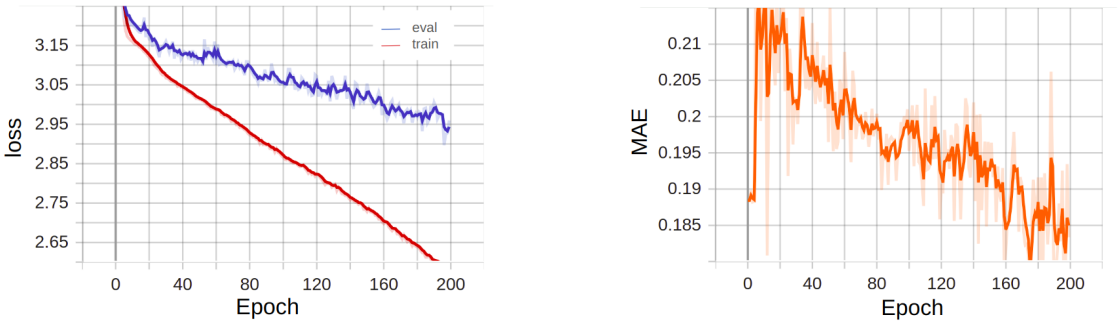

Figura 3.27: The left plot is training and validation loss trends for JSD over 200 epochs with aggregation layers for DINOv2. The right plot shows the MAE trend during training.

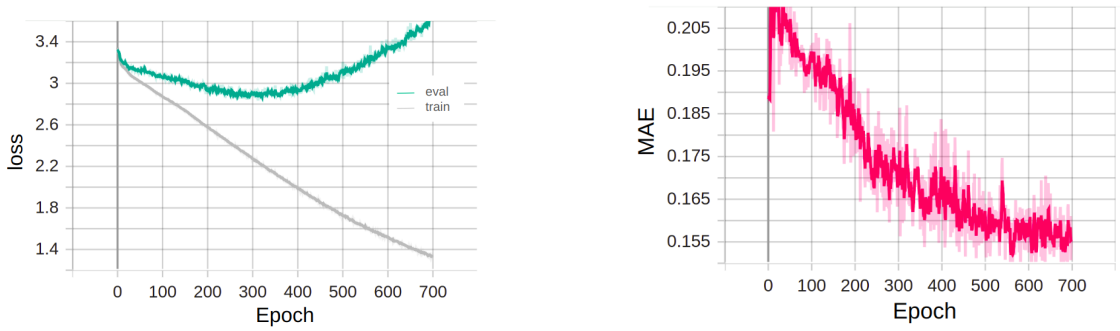

Figura 3.28: The left plot is training and validation loss trends for JSD over 700 epochs with aggregation layers for DINOv2. The right plot shows the MAE trend during training.

- The training can continue to epoch 300 without overfit.

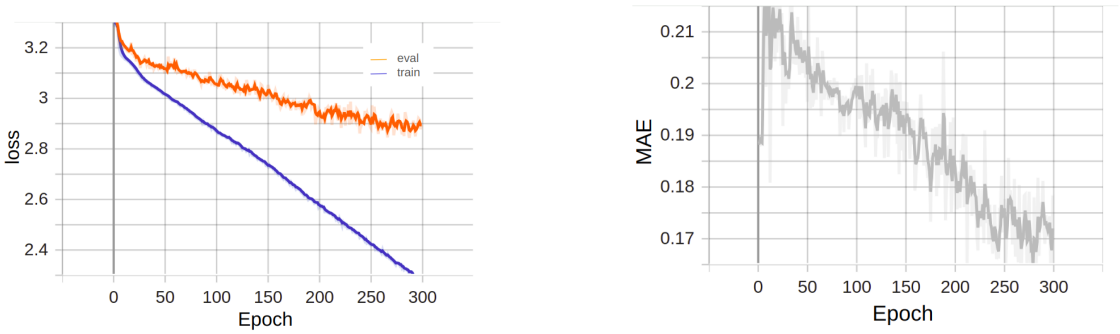

Figura 3.29: The left plot is training and validation loss trends for JSD over 300 epochs with aggregation layers for DINOv2. The right plot shows the MAE trend during training.

change in learning rate and weight decay

- weight decay is set to 1e-5 and the learning rate is initially set to 1e-4 and is reduced by a factor of 0.5 with the patience of 15 steps based on evaluation loss. For figure 3.30 the stopped epoch is 159.

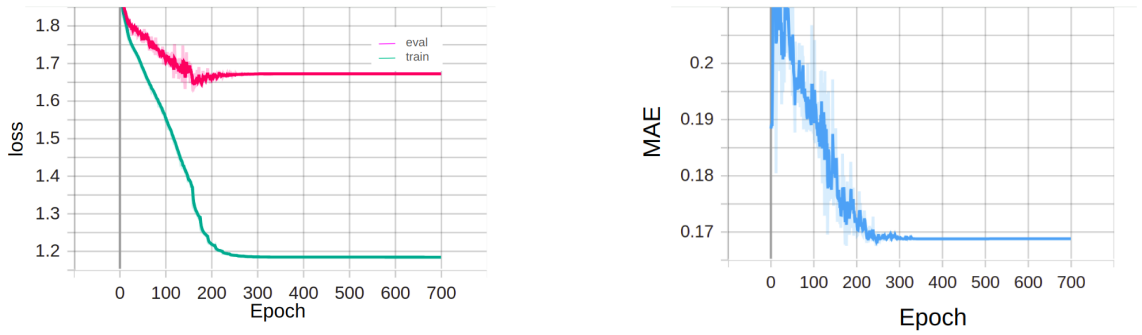

Figura 3.30: The left plot shows training and validation loss trends for JSD over 700 epochs with aggregation layers and with learning rate and weight decay adjustments. The right plot shows the MAE trend during training.

- weight decay is set to 1e-5 and the learning rate is initially set to 1e-4 and is reduced by a factor of 0.5 with the patience of 50 steps based on evaluation loss. For figure 3.31 the stopped epoch is 223.

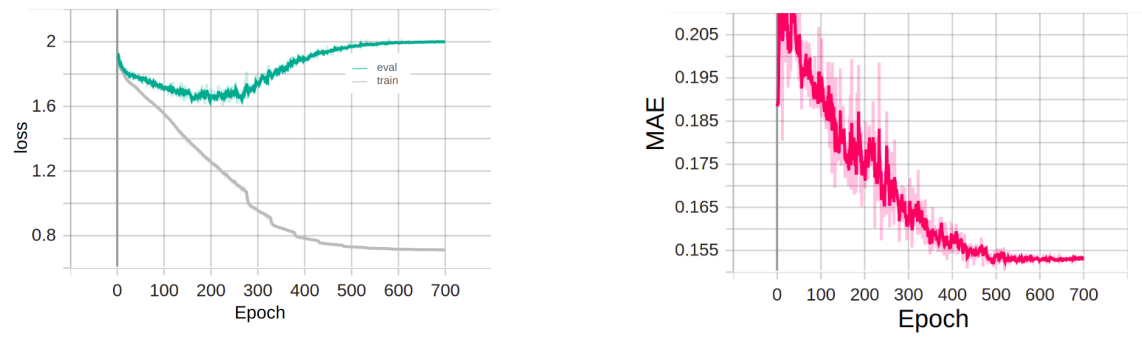

Figura 3.31: The left plot shows training and validation loss trends for JSD over 700 epochs with aggregation layers and with learning rate and weight decay adjustments. The right plot shows the MAE trend during training.

3. **Cross-Entropy (CE) Loss**

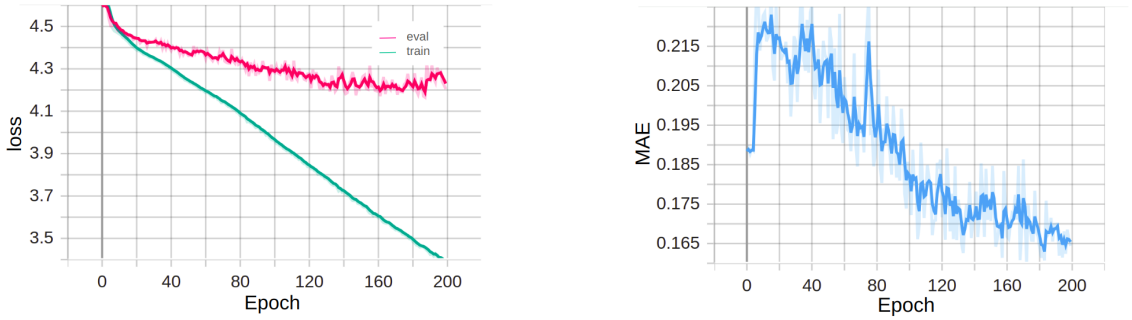

Figura 3.32: The left plot is training and validation loss trends for CE over 200 epochs with aggregation layers for DINOv2. The right plot shows the MAE trend during training.

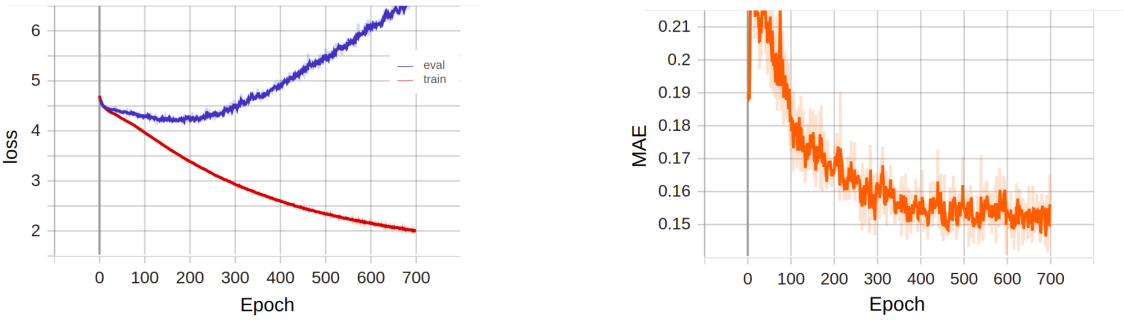

Figura 3.33: The left plot is training and validation loss trends for CE over 700 epochs with aggregation layers and with learning rate and weight decay adjustments. The right plot shows the MAE trend during training.

change in learning rate and weight decay

- weight decay is set to 1e-5 and the learning rate was initially set to 1e-4 and was reduced by a factor of 0.75 with the patience of 15 steps based on evaluation loss. For figure 3.34 the stopped epoch is 187.

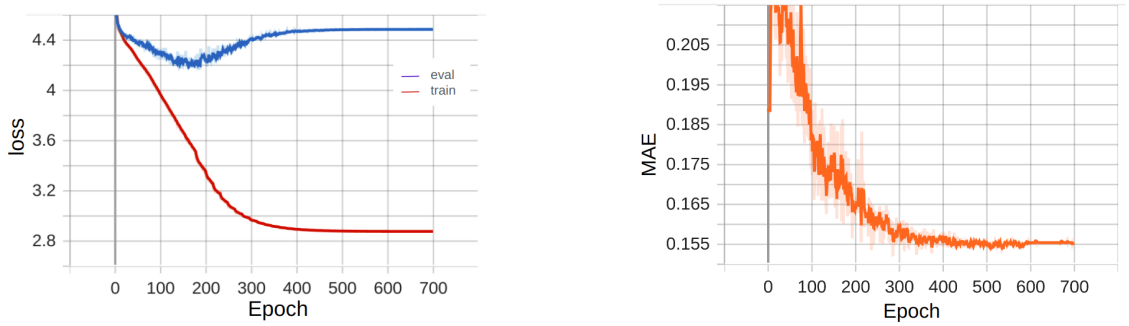

Figura 3.34: The left plot shows training and validation loss trends for CE over 700 epochs with aggregation layers and with learning rate and weight decay adjustments. The right plot shows the MAE trend during training.

- weight decay is set to 1e-6 and the learning rate is initially set to 2e-4 and is reduced by a factor of 0.75 with the patience of 15 steps based on evaluation loss. For figure 3.35 the stopped epoch is 98.

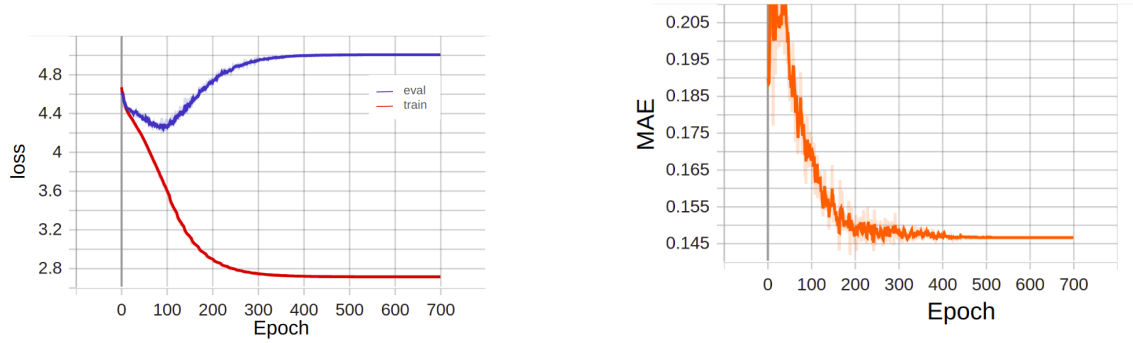

Figura 3.35: The left plot shows training and validation loss trends for CE over 700 epochs with aggregation layers and with learning rate and weight decay adjustments. The right plot shows the MAE trend during training.

Tabella 3.2: comparison between different configurations and loss functions, after implementing the aggregation.

Aggregation			known			Unknown		
loss func	epoch	accuracy	MAE	correlation	NLL	MAE	correlation	NLL
KLD	200	28%	160	0.234	6.66	**194**	0.09	12.27
KLD (WD+LR)	88	14%	203	0.230	6.76	204	0.03	10.094
	167	25%	162	0.233	6.67	215	0.08	11.87
	173	29%	**155**	0.233	6.73	228	0.09	12.63
JSD	200	21%	183	0.153	9.89	220	0.127	15.08
	300	26%	178	0.113	11.44	218	0.117	16.38
JSD (WD+LR)	159	20%	172	0.183	9.19	218	0.123	14.43
	223	27%	160	0.171	10.40	211	0.102	13.19
CrossEnt	200	26%	164	0.233	6.63	241	0.08	11.89
CrossEnt (WD+LR)	187	28%	162	0.228	6.41	218	0.09	11.71
	98	27%	166	0.233	6.72	211	0.08	11.95

In table 3.2, we compare all the evaluation metrics for known and unknown test sets for DINOv2+Aggregation.

Using aggregation for DINOv2, we tested all loss functions, finding that they performed well at 200 epochs but exhibited overfitting by 700 epochs. Despite various attempts to mitigate overfitting by adjusting learning rates (LR) and weight decay (WD), the DINOv2 model with KLD loss continued to overfit, with only slight improvements. Therefore, we employed early stopping, and the results are summarized in the table3.2. While the last configuration of KLD with WD and LR adjustments yielded the best result in terms of known MAE, the KLD with aggregation at 200 epochs showed the best MAE for the unknown test set. This indicates that although some configurations performed better for known data, early stopping at 200 epochs provided the best generalization for unknown data.

3.3.6 Optimizer Comparison for Best Configuration

To identify the most effective optimization strategy for the best training configuration among DONOv2 3.3.4 and DINOv2+aggregation 3.3.5, we conducted experiments using different optimizers. We focused on two configurations and tested several optimizers. We chose the best configuration based on the MAE for the unknown test which is the main objective of this thesis. For the DINOv2 part, Training with KLD loss function for 200 epochs and JSD loss function for 200 epochs have better result, and between these two the one with less MAE in known class was chosen. For DINOv2+Aggregation we chose the one with KLD loss function with the least Unknown MAE. Among all optimizers, RAdam and RMSprop emerged as the most promising. Here, we provide a brief overview of each optimizer and discuss their impact on the training process.

Rectified Adam (RAdam)

RAdam is an adaptive learning rate optimizer designed to address the slow convergence and instability issues associated with the Adam optimizer, especially in the early stages of training. RAdam combines the benefits of the adaptive learning rate and the rectified (adaptive) steps, which help accelerate the convergence and stabilize the training. Here we used the same learning rate (1e-4) and weight decay (1e-5). Betas is set to (0.9, 0.999) and eps is equal to 1e-8.

DINOv2

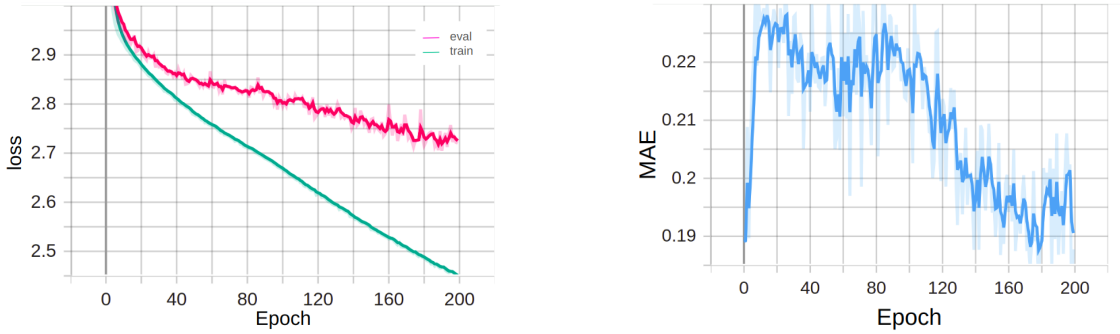

Figura 3.36: The left plot shows training and validation loss trends for KLD over 200 epochs for RAdam optimizer. The right plot shows the MAE trend during training.

DINOv2+Aggregation

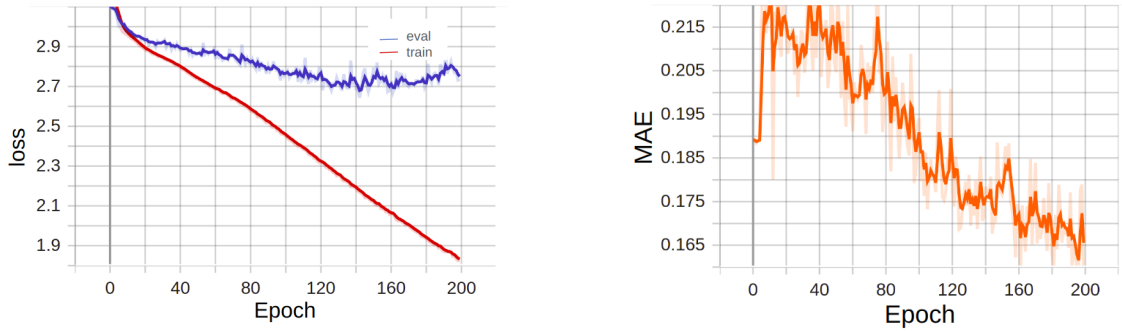

Figura 3.37: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for RAdam optimizer. The right plot shows the MAE trend during training.

Root Mean Square Propagation (RMSprop)

The second optimizer we tried is RMSprop. It is another adaptive learning rate method designed to maintain a moving average of the square of gradients and normalize the gradient step by this average. This helps in reducing the oscillations and speeding up the convergence, especially useful for neural networks and problems with noisy gradients. Here again, we used the same learning rate (1e-4) and weight decay (1e-5). Alpha is set to 0.99, momentum is 0.9 and eps is equal to 1e-08.

DINOv2

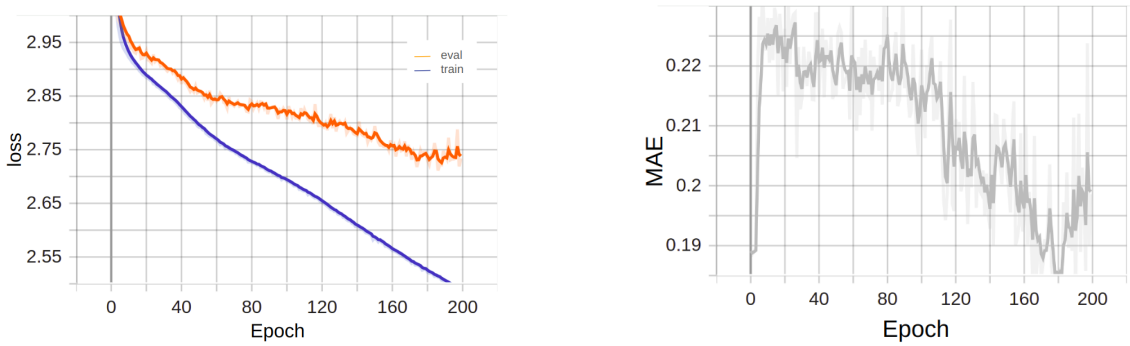

Figura 3.38: The left plot shows training and validation loss trends for KLD over 200 epochs for the RMSprop optimizer. The right plot shows the MAE trend during training.

DINOv2+Aggregation

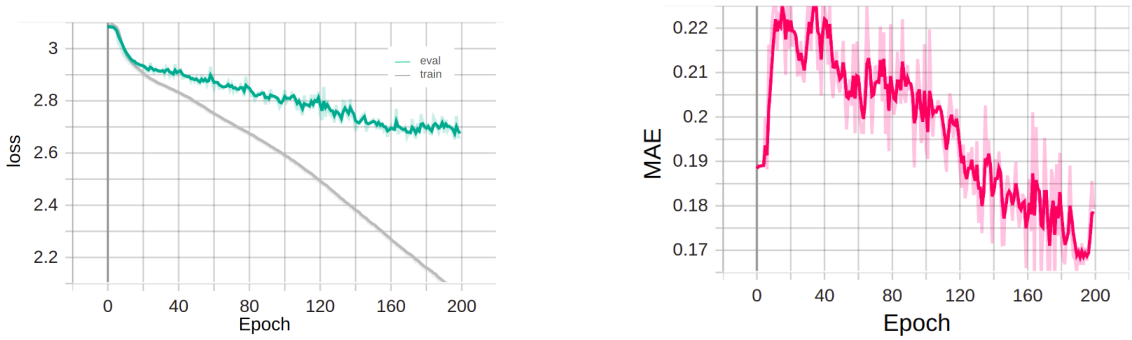

Figura 3.39: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RMSprop optimizer. The right plot shows the MAE trend during training.

Tabella 3.3: comparison between different optimizers before and after implementing the aggregation.

		known			Unknown		
optimizer	accuracy	MAE	correlation	NLL	MAE	correlation	NLL
Adam	22%	181	0.234	6.68	206	0.04	10.46
Adam+Agg	28%	160	0.234	6.66	194	0.09	12.27
RAdam	23%	187	0.233	6.67	211	0.04	10.51
RAdam+Agg	29%	**155**	0.234	6.34	**191**	0.08	12.15
RMSprop	22%	198	0.236	6.65	201	0.03	10.44
RMSprop+Agg	26%	179	0.242	10.58	227	0.03	12.76

In table 3.3, we compare all the evaluation metrics for known and unknown test sets for different optimizers. The configuration RAdam+Aggregation yielded the lowest MAE of 191 for the unknown test set, demonstrating the importance of fine-tuning optimizer parameters for enhanced model performance.

Fine tuning the parameters of RAdam

In our effort to optimize the RAdam parameters for enhancing model performance, we adopted a systematic approach, where we sequentially adjusted individual hyperparameters while keeping others constant. This stepwise tuning allowed us to identify the optimal configuration without exhaustively testing all possible combinations, which would be computationally prohibitive. The initial settings were as follows:

- learning rate: 1e-4
- weight decay: 1e-5
- betas: (0.9,0.999)
- epsilon: 1e-8

To determine the optimal learning rate, we tested three values: 1e-3, 1e-4, and 1e-5. Each learning rate was applied individually, and the model's performance was evaluated based on the improvement in Mean Absolute Error (MAE) and the overall convergence of the training and validation loss curves. The learning rate that yielded the best result in terms of minimizing MAE and ensuring stable convergence was then selected as the baseline for the subsequent tuning of other parameters. This stepwise approach allowed us to systematically optimize the model's hyperparameters, ensuring that each adjustment was based on the most effective configuration identified in the previous step.

After establishing the optimal learning rate, we proceeded to adjust the weight decay parameter to further fine-tune the model's performance. We tested the following values: 1e-4, 1e-5, and 1e-6. Each weight decay value was evaluated for its impact on reducing overfitting and enhancing the model's generalization capabilities. The weight decay parameter that demonstrated superior results in terms of lowering MAE and improving the stability of the validation loss was selected for the next phase of tuning.

With the optimal learning rate and weight decay determined, we focused on adjusting the beta parameters, which influence the momentum and variance in the optimization process. We tested three configurations: betas=(0.85,0.995), betas=(0.9,0.999), and betas=(0.98,0.9995). Each configuration was assessed based on the model's performance improvements and its ability to converge effectively. The betas values that led to the best overall performance, characterized by the lowest MAE and stable training dynamics, were chosen for the final parameter adjustment.

Finally, the epsilon value, which controls numerical stability, was adjusted. We tested the following values: 1e-7, 1e-8, 1e-9. The epsilon value that provided the best results was identified and used in the final optimizer configuration.

The results from these experiments are shown below:

learning rate:1e-3 learning rate:1e-5

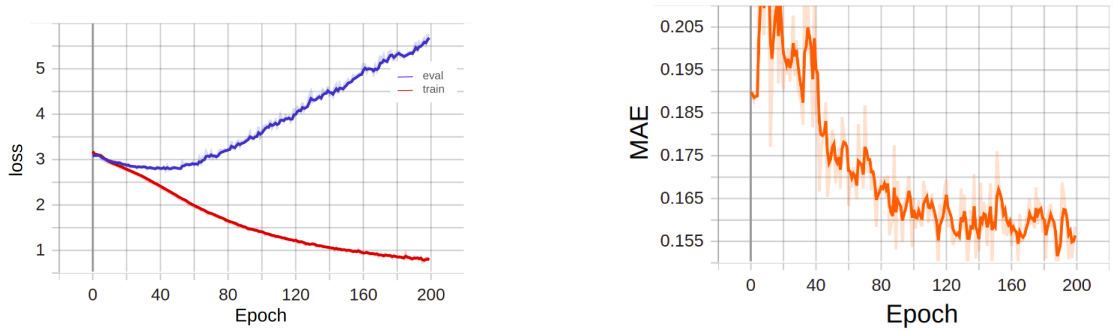

Figura 3.40: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the learning rate. The right plot shows the MAE trend during training.

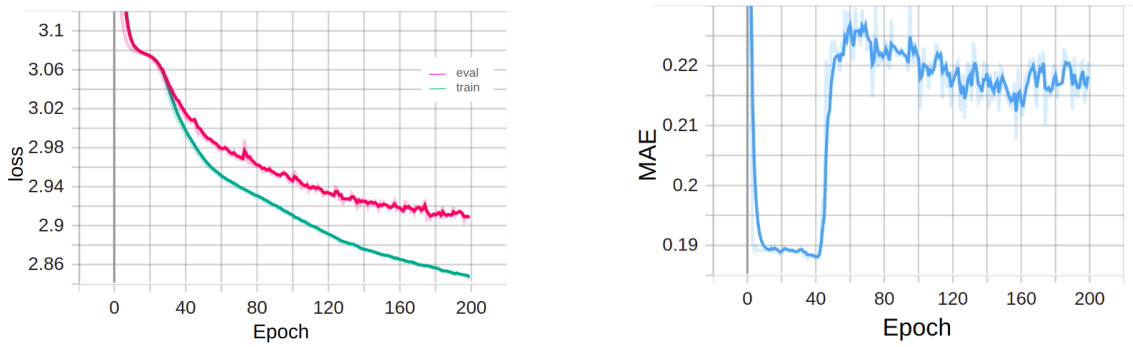

Figura 3.41: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the learning rate. The right plot shows the MAE trend during training.

Since the training with learning rate = 1e-3 was observed as overfit, we just compare the result of learning rate equal to 1e-4 and 1e-5. The MAE of unknown test for the first one is 191 and for the second one is 238. Therefore we chose 1e-4 for the learning rate.

weight decay:1e-4

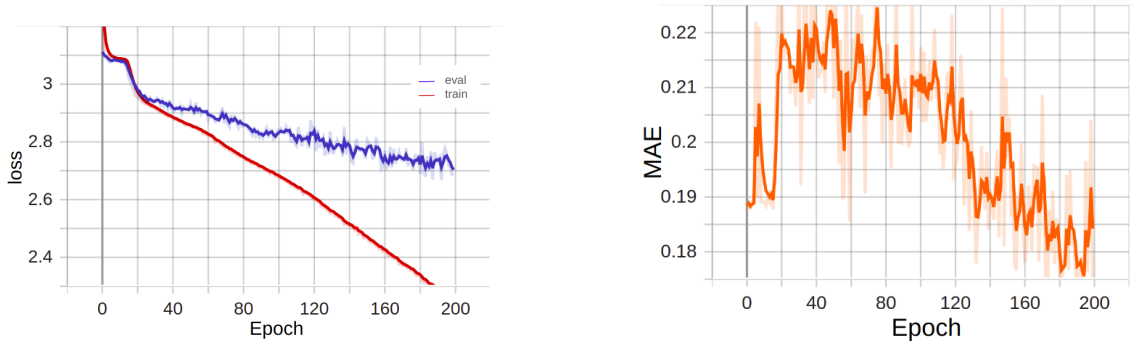

Figura 3.42: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the weight decay. The right plot shows the MAE trend during training.

weight decay:1e-6

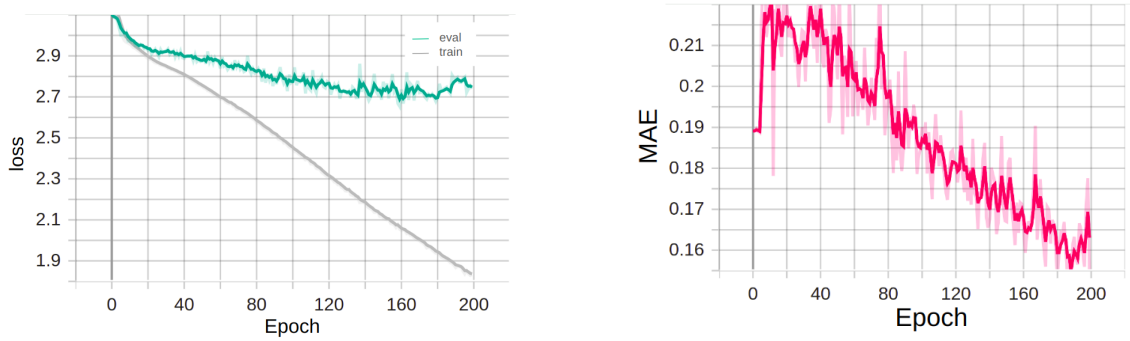

Figura 3.43: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the weight decay. The right plot shows the MAE trend during training.

The MAE of the unknown test for weight decay of 1e-4, 1e-5, and 1e-6 are respectively, 213,191,190. Therefore we chose 1e-6 for the weight decay.

Betas:(0.85,0.995)

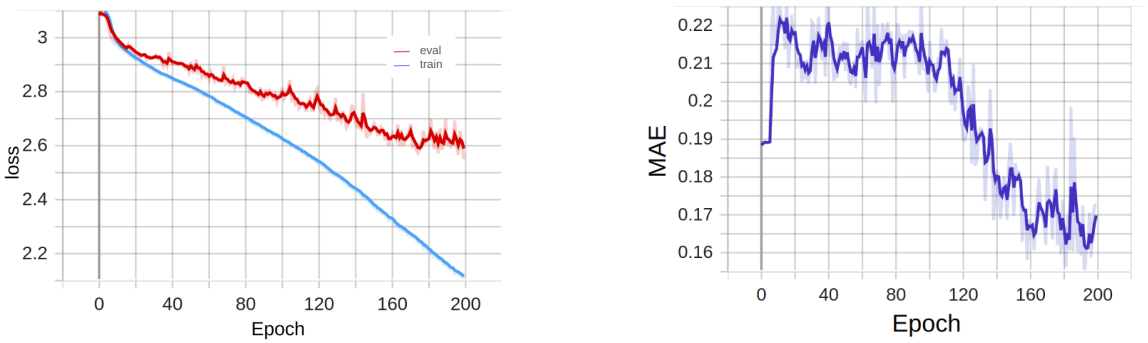

Figura 3.44: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the betas. The right plot shows the MAE trend during training.

Betas:(0.98,0.9995)

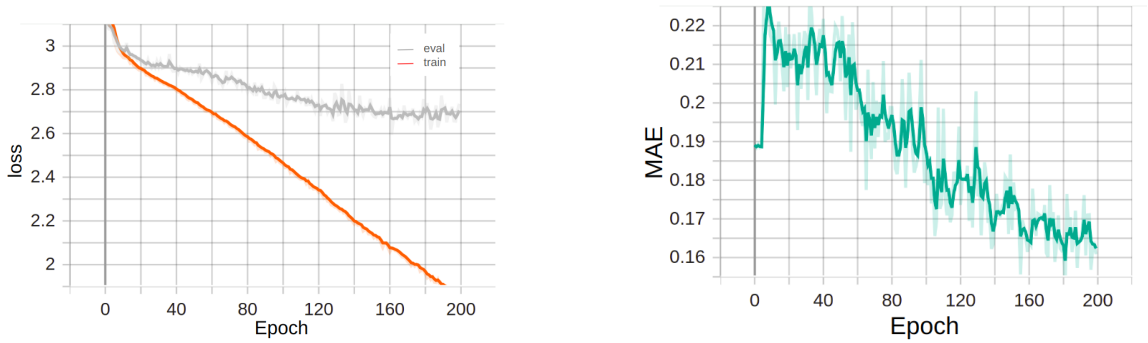

Figura 3.45: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the betas. The right plot shows the MAE trend during training.

The MAE of the unknown test for betas (0.85,0.995),(0.9,0.999), and (0.98,0.9995) are respectively, 199,190,242. Therefore we chose (0.9,0.999) for the weight decay.

epsilon:1e-7

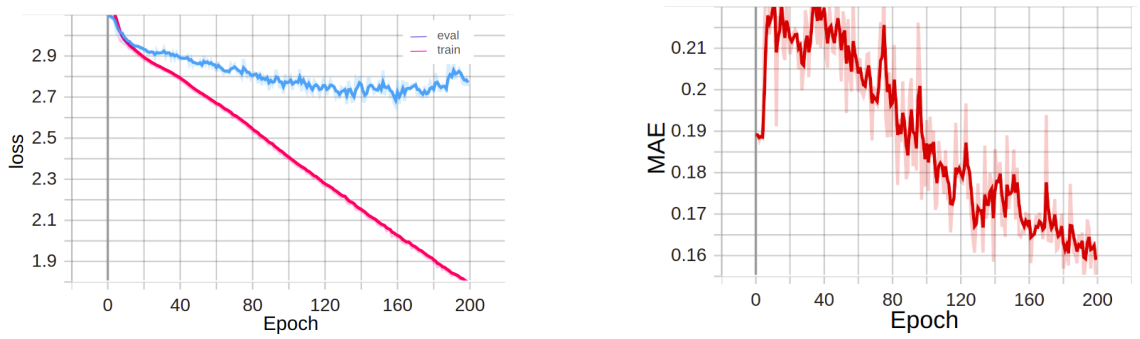

Figura 3.46: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the epsilon. The right plot shows the MAE trend during training.

espsilon:1e-9

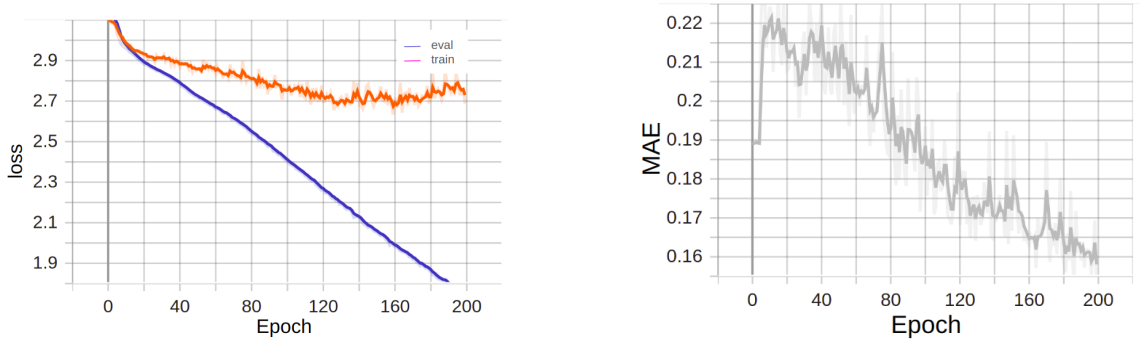

Figura 3.47: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with changing the epsilon. The right plot shows the MAE trend during training.

The MAE of the unknown test for betas 1e-7,1e-8, and 1e-9 are respectively, 206,190,200. Therefore we chose 1e-8 for the weight decay.

Tabella 3.4: Fine tuning the hyperparameters of RAdam

LR	WD	betas	eps	MAE known	MAE unknown
1e-3				overfit	overfit
1e-4	1e-5	(0.9,0.999)	1e-8	155	191
1e-5				220	238
	1e-4			172	213
1e-4	1e-5	(0.9,0.999)	1e-8	155	191
	1e-6			153	190
		(0.85,0.995)		172	199
1e-4	1e-6	(0.9,0.999)	1e-8	153	190
		(0.98,0.9995)		160	242
			1e-7	154	206
1e-4	1e-6	(0.9,0.999)	1e-8	153	**190**
			1e-9	149	200

Augmentation

Feature Augmentation: For the best configuration we have had in the last step, we decided to apply the same feature augmentation here as well, but there is an overfit in the result.

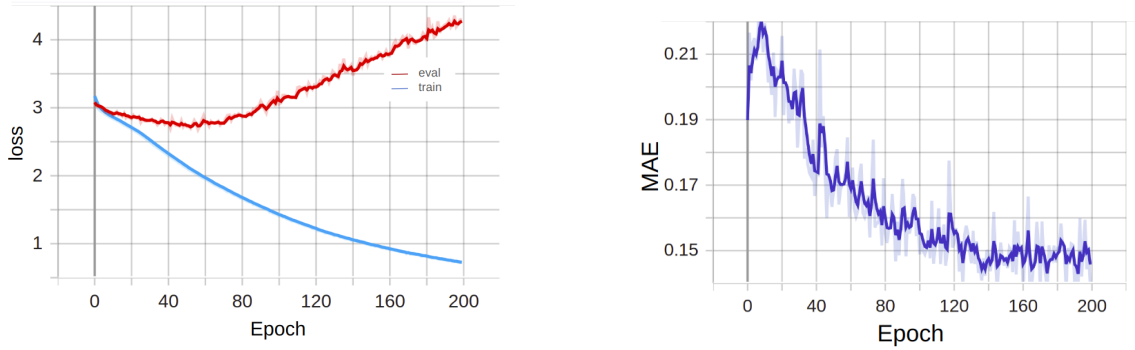

Figura 3.48: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with data Augmentation. The right plot shows the MAE trend during training.

We also tried to use less variant in noise and drop out (gaussian noise with mean = 0 and std = 0.02 and drop out rate = 0.05). We again observe the overfit.

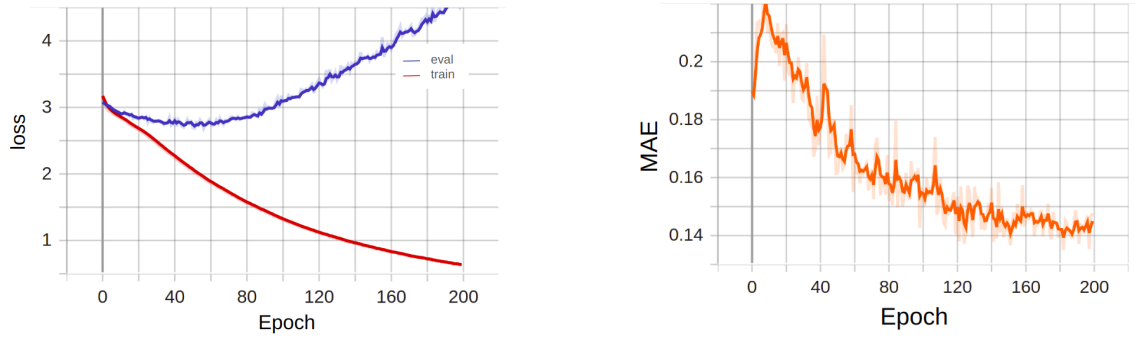

Figura 3.49: The left plot shows training and validation loss trends for KLD over 200 epochs with aggregation layers for the RAdam optimizer with data Augmentation. The right plot shows the MAE trend during training.

In our study, overfitting was observed following feature-level data augmentation, despite the absence of overfitting in the original dataset. This phenomenon

can be attributed to several factors. First, the augmentation techniques may have introduced unrealistic variations or noise into the feature space, which the model overfitted rather than learning robust, generalizable patterns. Additionally, over-augmentation or the use of techniques that do not align well with the inherent characteristics of the features might have shifted the data distribution, making the augmented features less representative of the original dataset. This can lead the model to learn features specific to the augmented data, reducing its generalization ability to unseen data. Finally, the regularization parameters optimized for the original data might have been insufficient for the augmented dataset, exacerbating overfitting. Consequently, while augmentation aims to enhance generalization, the inappropriate implementation or choice of augmentation methods can instead introduce overfitting.

Distribution of Predicted vs. Ground Truth Caloric Values

To assess the performance of our model in predicting caloric values, we compared the predicted distributions with the ground truth distributions for both known and unknown test sets. The figures below show these comparisons under different configurations and optimizers.

Figure 3.50 (a) and (b) represent the distribution plots for the known and unknown test results respectively, using the Adam optimizer without any learning rate and weight decay adjustments. The blue line represents the predicted distribution, while the orange line represents the ground truth distribution. These initial attempts indicate that there is a discrepancy between the predicted and actual caloric values, especially evident in the divergence of the blue and orange lines.

Figure 3.50 (c) and (d) illustrate the results of using the RAdam optimizer with the best configuration for hyperparameters, including a KLD loss function over 200 epochs with aggregation. In these plots, we observe a significant improvement in the alignment between the predicted (blue line) and ground truth (orange line) distributions. This indicates that the RAdam optimizer with the selected hyperparameters provides a more accurate prediction of caloric values compared to the Adam optimizer without adjustments.

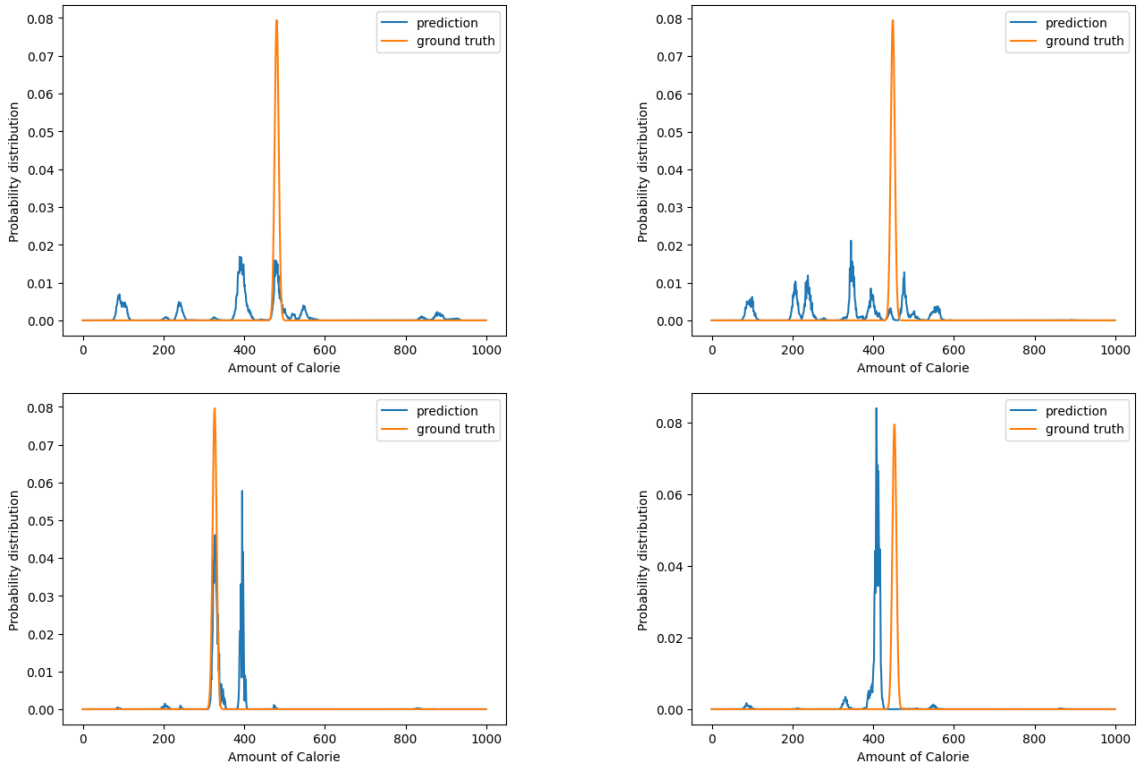

Figura 3.50: (a) distribution plots for the known test, (b) distribution plots for the unknown test, both using the Adam optimizer without any learning rate and weight decay adjustments, (c) distribution plots for the known test, (d) distribution plots for the unknown test, using the RAdam optimizer with the best configuration for hyperparameters.

Capitolo 4

Discussion and Conclusion

4.1 Discussion

The primary objective of this thesis was to develop an advanced, accurate, and generalizable model for estimating caloric expenditure from video data, leveraging pre-trained foundation models, and exploring various loss functions to enhance model performance. Our experiments provided valuable insights into the complexities and challenges associated with this task.

4.1.1 Challenges with CLIP Features

In our initial experiments using features extracted from the CLIP model, we encountered significant overfitting across all loss functions (KLD, JSD, and CE). Even with careful changes to the learning rate and weight decay, overfitting continued. This suggests that CLIP's strong feature extraction might be too powerful, hiding the subtle details needed for accurate caloric estimation from video data. These findings highlighted the necessity for more sophisticated regularization techniques or alternative loss functions tailored to handle the intricacies of features derived from advanced models like CLIP.

Efforts to Mitigate Overfitting

To address the persistent overfitting, we implemented a combination of feature augmentation, video augmentation, and further tuning of learning rate and

weight decay. Feature augmentation techniques, including the addition of Gaussian noise and feature dropout, aimed to enhance feature robustness, while video augmentation increased training data variability. However, these strategies failed to resolve the overfitting effectively. Adjustments to learning rate and weight decay were similarly unproductive; either failing to regularize the model sufficiently or overly constraining it, preventing effective learning.

4.1.2 Exploring DINOv2

We then explored using DINOv2 for feature extraction, evaluating the model's performance across KLD, JSD, and CE loss functions. Initial training for 200 and 700 epochs demonstrated commendable performance across all models. While the Cross-Entropy loss function yielded the lowest MAE for the known test set at 700 epochs, it did not generalize well to the unknown test set. Conversely, KLD and JSD losses at 200 epochs outperformed others on the unknown test set, with KLD showing superior results in both known and unknown MAE.

Aggregation and Overfitting

Using aggregation for DINOv2, we observed that models performed well at 200 epochs but overfitting persisted for 700 epochs. Despite various learning rate and weight decay adjustments, overfitting continued, though with slight improvements. We employed early stopping, which provided better generalization for unknown data, as summarized in Table 3.2. While KLD with aggregation at 200 epochs yielded the best MAE for the unknown test set, configurations with higher epochs often overfitted.

4.1.3 Optimization Strategies

We identified RAdam and RMSprop as the most promising optimizers for the best training configuration among DINOv2 and DINOv2+aggregation. The configuration using RAdam with aggregation yielded the lowest MAE of 191 for the unknown test set, highlighting the importance of fine-tuning optimizer parameters for enhanced model performance.

Parameter Tuning for RAdam

To optimize RAdam parameters, we systematically adjusted individual hyperparameters. This approach, focusing on learning rate, weight decay, betas, and epsilon values, enabled us to identify an optimal configuration without exhaustively testing all combinations. Despite the refined parameter settings, overfitting persisted when feature augmentation was applied, indicating the need for further research into more effective augmentation strategies and regularization techniques.

Performance Assessment

To assess model performance, we compared the predicted distributions with ground truth distributions for both known and unknown test sets. Figures 3.50(a) and 3.50(b) illustrate the discrepancies between predicted and actual values using Adam optimizer without adjustments. In contrast, Figures 3.50(c) and 3.50(d) demonstrate significant improvement using RAdam with the best configuration, indicating a more accurate prediction of caloric values.

4.2 Conclusion

This thesis underscores the challenges of estimating caloric expenditure from video data, highlighting the limitations of traditional methods and the potential of foundation models. Despite the robust feature extraction capabilities of models like CLIP and DINOv2, overfitting remains a critical issue, necessitating sophisticated regularization techniques and tailored loss functions.

Our experiments demonstrated that while foundation models provide a solid starting point, achieving optimal performance requires careful tuning of hyperparameters, effective augmentation strategies, and the use of advanced optimizers. The promising results with RAdam optimizer and early stopping suggest a path forward for future research.

In conclusion, this work advances the understanding of video-based caloric expenditure estimation, offering valuable insights and practical methodologies for developing more accurate and generalizable models. Future research should continue exploring novel regularization techniques, diverse datasets, and innovative model architectures to further enhance the accuracy and applicability of these models in real-world scenarios.

Bibliography

[1] Di Wu, Nabin Sharma, and Michael Blumenstein. "Recent advances in video-based human action recognition using deep learning: A review". In: *2017 International Joint Conference on Neural Networks (IJCNN)* (2017).

[2] Hieu H. Pham et al. "Video-based Human Action Recognition using Deep Learning: A Review". In: *arXiv:2208.03775* (2022).

[3] Zehua Sun et al. "Human Action Recognition From Various Data Modalities: A Review". In: *IEEE Transactions on Pattern Analysis and Machine Intelligence* 45.3 (2023), pp. 3200–3225. DOI: 10.1109/TPAMI.2022.3183112.

[4] H. Lee, R Grosse R.and Ranganath, and A. Y. Ng. "Convolutional deep belief networks for scalable unsupervised learning of hierarchical representations". In: *ICML, pp. 609–616* (2009).

[5] Thippa Reddy Gadekallu et al. "Hand gesture recognition based on a Harris Hawks optimized Convolution Neural Network". In: *Computers and Electrical Engineering* (2022).

[6] Joao Carreira and Andrew Zisserman. *Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset*. 2018. arXiv: 1705.07750 [cs.CV].

[7] Arman J. Paydarfar, Antonio Prado, and Sunil K. Agrawal. "Human Activity Recognition Using Recurrent Neural Network Classifiers on Raw Signals from Insole Piezoresistors". In: *2020 8th IEEE RAS/EMBS International Conference for Biomedical Robotics and Biomechatronics (BioRob)*. 2020, pp. 916–921. DOI: 10.1109/BioRob49111.2020.9224311.

[8] Schalk Wilhelm Pienaar and Reza Malekian. "Human Activity Recognition using LSTM-RNN Deep Neural Network Architecture". In: *2019 IEEE 2nd Wireless Africa Conference (WAC)*. 2019, pp. 1–5. DOI: 10.1109/AFRICA.2019.8843403.

[9] Siqi Liu, Nan Wu, and Haifeng Jin. "Human Action Recognition Based on Attention Mechanism and HRNet". In: *Proceeding of 2021 International Conference on Wireless Communications, Networking and Applications*. Ed. by Zhihong Qian, M.A. Jabbar, and Xiaolong Li. Singapore: Springer Nature Singapore, 2022, pp. 279–291. ISBN: 978-981-19-2456-9.

[10] Sannara Ek, François Portet, and Philippe Lalanda. "Transformer-based models to deal with heterogeneous environments in Human Activity Recognition". In: *Personal and Ubiquitous Computing* 27.6 (Nov. 2023), pp. 2267–2280. ISSN: 1617-4917. DOI: 10.1007/s00779-023-01776-3. URL: http://dx.doi.org/10.1007/s00779-023-01776-3.

[11] Will Kay et al. *The Kinetics Human Action Video Dataset*. 2017. arXiv: 1705.06950 [cs.CV].

[12] Fabian Caba Heilbron et al. "ActivityNet: A large-scale video benchmark for human activity understanding". In: *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. 2015, pp. 961–970. DOI: 10.1109/CVPR.2015.7298698.

[13] Alina Roitberg et al. *Uncertainty-sensitive Activity Recognition: a Reliability Benchmark and the CARING Models*. 2021. arXiv: 2101.00468 [cs.CV].

[14] Kunyu Peng et al. *Should I take a walk? Estimating Energy Expenditure from Video Data*. 2022. arXiv: 2202.00712 [cs.CV].

[15] Fahd Albinali et al. "Using wearable activity type detection to improve physical activity energy expenditure estimation". In: *UbiComp* (2010).

[16] Onur Barut, Li Zhou, and Yan Luo. "Multitask LSTM model for human activity recognition and intensity estimation using wearable sensor data". In: *EEE Internet of Things Journal* (2020).

[17] Mathias Hedegaard et al. "Prediction of energy expenditure during activities of daily living by a wearable set of inertial sensors".

[18] Bradley Kendall, Bryanne Bellovary, and Neha P. Gothe. "Validity of wearable activity monitors for tracking steps and estimating energy expenditure during a graded maximal treadmill test". In: *Journal of Sports Sciences* (2019).

[19] Gunnar Farnebäck. "Two-Frame Motion Estimation Based on Polynomial Expansion". In: vol. 2749. June 2003, pp. 363–370. ISBN: 978-3-540-40601-3. DOI: 10.1007/3-540-45103-X_50.

[20] John Barron, David Fleet, and S. Beauchemin. "Performance Of Optical Flow Techniques". In: *International Journal of Computer Vision* 12 (Feb. 1994), pp. 43–77. DOI: 10.1007/BF01420984.

[21] Lijuan Zhou et al. *Human Pose-based Estimation, Tracking and Action Recognition with Deep Learning: A Survey*. 2023. arXiv: 2310.13039 [id='cs.CV' full$_n$ame =′ ComputerVisionandPatternRecognition′is$_a$ctive = Truealt$_n$ame = Nonein$_a$rchive =′ cs′is$_g$eneral = Falsedescription =′ Coversimageprocessing, computervision, patt

[22] Zhe Cao et al. *OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*. 2019. arXiv: 1812.08008 [cs.CV].

[23] Hao-Shu Fang et al. *AlphaPose: Whole-Body Regional Multi-Person Pose Estimation and Tracking in Real-Time*. 2022. arXiv: 2211.03375 [cs.CV].

[24] Sijie Yan, Yuanjun Xiong, and Dahua Lin. "Spatial Temporal Graph Convolutional Networks for Skeleton-Based Action Recognition". In: *Proceedings of the AAAI Conference on Artificial Intelligence* 32 (Jan. 2018). DOI: 10.1609/aaai.v32i1.12328.

[25] Zihan Wang et al. *Deep Neural Networks in Video Human Action Recognition: A Review*. 2023. arXiv: 2305.15692 [cs.CV].

[26] Georgia Gkioxari et al. *R-CNNs for Pose Estimation and Action Detection*. 2014. arXiv: 1406.5212 [cs.CV].

[27] Mathilde Caron et al. *Emerging Properties in Self-Supervised Vision Transformers*. 2021. arXiv: 2104.14294 [cs.CV].

[28] Alec Radford et al. *Learning Transferable Visual Models From Natural Language Supervision*. 2021. arXiv: 2103.00020 [cs.CV].

[29] Johannes Schneider, Christian Meske, and Pauline Kuss. "Foundation Models". In: *Business Information Systems Engineering* 66 (Jan. 2024), pp. 1–11. DOI: 10.1007/s12599-024-00851-0.

[30] Liuqing Chen, Lingyun Sun, and Ji Han. "A Comparison Study of Human and Machine-Generated Creativity". In: *Journal of Computing and Information Science in Engineering* 23.5 (Apr. 2023), p. 051012. ISSN: 1530-9827. DOI: 10.1115/1.4062232. eprint: https://asmedigitalcollection.asme.org/computingengineering/article-pdf/23/5/051012/7001808/jcise_23_5_051012.pdf. URL: https://doi.org/10.1115/1.4062232.

[31] Maxime Oquab et al. *DINOv2: Learning Robust Visual Features without Supervision*. 2024. arXiv: 2304.07193 [cs.CV].

[32] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. *UCF101: A Dataset of 101 Human Actions Classes From Videos in The Wild*. 2012. arXiv: 1212.0402 [cs.CV].

[33] Hilde Kuehne et al. "HMDB51: A Large Video Database for Human Motion Recognition". In: Nov. 2011, pp. 2556–2563. ISBN: 978-3-642-33373-6. DOI: 10.1109/ICCV.2011.6126543.

[34] Barbara E. Ainsworth et al. "2011 Compendium of Physical Activities: A Second Update of Codes and MET Values". In: *Medicine and Science in Sports and Exercise* (2011).

[35] Hao-Shu Fang et al. *RMPE: Regional Multi-person Pose Estimation*. 2018. arXiv: 1612.00137 [cs.CV].

[36] Jiefeng Li et al. *CrowdPose: Efficient Crowded Scenes Pose Estimation and A New Benchmark*. 2019. arXiv: 1812.00324 [cs.CV].

[37] Yuliang Xiu et al. *Pose Flow: Efficient Online Pose Tracking*. 2018. arXiv: 1802.00977 [cs.CV].

[38] Pei-Fu Tsou and Chao-Cheng Wu. "Estimation of Calories Consumption for Aerobics Using Kinect Based Skeleton Tracking". In: *2015 IEEE International Conference on Systems, Man, and Cybernetics* (2015), pp. 1221–1226. URL: https://api.semanticscholar.org/CorpusID:27479384.

[39] Terrance DeVries and Graham W. Taylor. *Improved Regularization of Convolutional Neural Networks with Cutout*. 2017. arXiv: 1708.04552 [cs.CV].

[40] Deepak Pathak et al. *Context Encoders: Feature Learning by Inpainting*. 2016. arXiv: 1604.07379 [cs.CV].

[41] Gaoshuang Huang et al. *DINO-Mix: Enhancing Visual Place Recognition with Foundational Vision Model and Feature Mixing*. 2023. arXiv: 2311.00230 [cs.CV].

[42] Igor Iashin. *Video Features*. https://github.com/v-iashin/video_features. Accessed: 2024-06-10. 2021.

[43] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". In: *Neural Networks* 2.5 (1989), pp. 359–366. ISSN: 0893-6080. DOI: https://doi.org/10.1016/0893-6080(89)90020-8. URL: https://www.sciencedirect.com/science/article/pii/0893608089900208.

[44] Masahito Togami et al. "Unsupervised Training for Deep Speech Source Separation with Kullback-Leibler Divergence Based Probabilistic Loss Function". In: *ICASSP 2020 - 2020 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*. 2020, pp. 56–60. DOI: 10.1109/ICASSP40776.2020.9054171.

[45] Taehyeon Kim et al. *Comparing Kullback-Leibler Divergence and Mean Squared Error Loss in Knowledge Distillation*. 2021. arXiv: 2105.08919 [cs.LG].

[46] Erik Englesson and Hossein Azizpour. "Generalized Jensen-Shannon Divergence Loss for Learning with Noisy Labels". In: *Advances in Neural Information Processing Systems*. Ed. by M. Ranzato et al. Vol. 34. Curran Associates, Inc., 2021, pp. 30284–30297. URL: https://proceedings.neurips.cc/paper_files/paper/2021/file/fe2d010308a6b3799a3d9c728ee74244-Paper.pdf.

[47] Sebastian Ruder. *An Overview of Multi-Task Learning in Deep Neural Networks*. 2017. arXiv: 1706.05098 [cs.LG].

Acknowledgements

I would like to express my deepest gratitude to everyone who supported me throughout this thesis journey.
First and foremost, I am profoundly grateful to Professor Roitberg at the University of Stuttgart for her invaluable guidance, support, and mentorship during my internship and the completion of this thesis. Her expertise and insights have been instrumental in shaping this work.
I would also like to thank the faculty at the University of Padova for providing me with a solid academic foundation throughout my master's program. Special thanks go to my academic advisor at the University of Padova, Professor Zanetti, whose encouragement and feedback have been greatly appreciated.
A heartfelt thank you to my colleagues at the University of Padova, who have provided me with not only academic support but also friendship and encouragement during this journey.