June 2020

Action Recognition Using the Motion Taxonomy

Maxat Alibayev
University of South Florida

Action Recognition Using the Motion Taxonomy

by

Maxat Alibayev

A thesis submitted in partial fulfillment
of the requirements for the degree of
Master of Science in Computer Science
Department of Computer Science and Engineering
College of Engineering
University of South Florida

Major Professor: Yu Sun, Ph.D.
Dmitry B. Goldgof, Ph.D.
Sudeep Sarkar, Ph.D.

Date of Approval:
June 17, 2020

Keywords: Motion codes, Deep Learning, Embedding

Dedication

I dedicate my thesis to my wife, who was always there to help me and inspired me to explore and research new things. I also dedicate it to my parents and friends.

Acknowledgments

I would like to thank my major professor, Dr. Yu Sun, for teaching and guiding me throughout my research. I will always be grateful for the knowledge and experience I gained from working in his lab and taking his Deep Learning courses. I also want to thank my committee members, Dr. Dmitry Goldgof and Dr. Sudeep Sarkar. They introduced me to the world of image processing and computer vision, which, as a result, inspired me to do my research. And of course, I want to thank all my labmates. It was always a pleasure to come to the lab and work with these people. I should express special thanks to Ahmad Jelodar and David Paulius. Ahmad helped me join the team and was like a mentor to me, and David shared all his work with me, which became a crucial part of my research.

Table of Contents

List of Tables

List of Figures

Abstract

In the last years, modern action recognition frameworks with deep architectures have achieved impressive results on the large-scale activity datasets. All state-of-the-art models share one common attribute: two-stream architectures. One deep model takes RGB frames, while the other model is fed with pre-computed optical flow vectors. The outputs of both models are combined to be used as a final probability distribution for the action classes. When comparing the results of individual models with the fused model, it is common to see that that latter method is more superior. Researchers explain that phenomena with the fact that optical flow vectors serve as the low-level motion features.

With idea of representing motion features in a more explainable way, we develop a motion prediction framework that extracts high-level motion features from videos represented as the binary *motion codes*. We derive the motion codes from the *motion taxonomy*, a hierarchical structure that defines salient motion attributes. We also integrate the extracted motion features into the state-of-the-art action recognition model and achieve improved performance over the baseline model.

In addition to the motion representation, we develop a framework based on the cross-modal embedding concepts to learn an action recognition model that does not encode its labels with one-hot vectors. More specifically, we represent the narrated annotation words via embedded word vectors and learn to embed visual and text data into shared vector space. The resulting model eliminates the shortcomings of one-hot vectors and achieves performance competitive with conventional baselines on the coarse-grained action classification task.

Chapter 1: Introduction and Background

1.1 Overview

Roboticists for years have aimed for developing robots or intelligent agents for activities of daily living. One aspect of this field is the development of good motion representation that the robot could understand to detect and replicate human activities. Typically, humans use words to refer to their actions (e.g., take, cut, chop, mix, wipe). The well-known word embedding models, such as Word2Vec [1] and GloVe [2], can embed those words into continuous vectors that the machine can understand. However, single words are semantically and mechanically ambiguous. For instance, the word "open" most likely refers to the action of opening the door. On the other hand, it may also refer to the action of opening a heat, a tap, a bag, and many other actions that have different motions. In addition to that, the word "open" belongs to multiple parts of speech, which means it may refer to the state of the object rather than action. Such kind of semantic ambiguity of words can be eliminated with word representations that depend on the sentence context [3], but it does not eliminate the mechanical ambiguity of words. One way of solving this problem is adding a separate modality that would represent the motions from mechanical perspective. The combination of such a motion representation and semantic labels could potentially produce fine-grained action representation. The semantic labels would answer the question of what action is shown in the video, while motion representation would answer the question of how that action is executed.

Nonetheless, defining motion representation is a non-trivial task. It may involve many variables, such as the interaction between the objects, the motion trajectories, the directions and magnitudes of applied forces to name a few [4, 5]. It is crucial to identify distinguishable motion attributes, as well as to structurally define them. It is also important to consider the context. For instance, the motions in the sports events and the motions in the common daily activities will have non-relevant attributes. Despite the mentioned difficulties, well-defined motion attributes can provide us with well-defined attribute space that can be used in other tasks, such as visual recognition [6, 7, 8, 9, 10]. Another benefit of such representation is the ability to teach the robot to execute the given actions. For instance, the action *cut the onion* can be done in multiple ways. Hence, the robot does not know how exactly to cut it. It may cut it once into a half, or it may cut it in small pieces. However, with an accurate knowledge about how to execute this task, along with the state recognition algorithms, the robot will be able to precisely replicate this action.

A well-defined motion representation can be also applied in building a knowledge representation [11], such as *functional object-oriented network* (FOON) [12, 13]. FOON is a knowledge representation that is built from observing human-object manipulations. It can be utilized in representing object-motion affordance [14, 15, 16], understanding long activities from videos [17], and knowledge retrieval for the robot. The structure of FOON can be illustrated as the network of *functional units*, each of which represents one atomic action. The atomic action consists from input objects and their states, the manipulation node, and the output objects and their states. The manipulation nodes are represented as verbs. After retrieving a functional unit, the robot knows its input objects and their states, and similarly for the output objects. However, it may not know how to get from the input to the output by purely relying on the semantic representation for the manipulation. Therefore, a mechanically informative motion representation can highly benefit a knowledge representation structure like FOON.

Another area that depends on motion representation is action recognition from demonstration videos. Nowadays, the state-of-the-art action recognition methods employ the pre-computed optical flow vectors from the video frames [18, 19, 20, 21, 22, 23, 24]. These vectors represent the motion of the individual pixels. Hence, the researchers use them to compute the motion features in the videos. However, optical flow frames are very high-dimensional and low-level features. It is hard to explain how exactly the deep models leverage them during the learning process. On the other side, if the model has a prior knowledge about low-dimensional and high-level motion features, it would require much less effort to recognize certain actions.

In addition to motion representation, modern action recognition frameworks have certain limitations associated with their semantic labels. The conventional method for supervised action recognition training is to label the input videos with action labels (e.g., nouns and verbs) that are represented as one-hot encoded vectors. A one-hot vector is an N-dimensional vector, where N is equal to the number of classes in the dataset. Its $N - 1$ dimensions are labeled with 0 and a single dimension is labeled as 1, whose index represents the class label. Such kind of label encoding demonstrated decent results in multi-class classification tasks and it is a relatively simple encoding strategy. However, this method has 2 major drawbacks. First of all, one-hot encoding suffers from high-cardinality datasets (i.e., datasets with too many classes) as the vector dimensionality increases linearly. This in turn results in grouping the semantic labels into coarse-grained classes. Second, the classes have no information about other classes. They do not now how similar or how different they are from other classes. If we use conventional distance or similarity metrics (e.g., Euclidean distance and cosine similarity), we will get identical values for each pair of unique categories.

1.2 Contributions and Outline

Considering the high demand for a well-defined motion representation, we developed a low-dimensional motion embedding for motions in cooking scenarios based on the *motion taxonomy* [25, 26] and train a deep model that predicts the motion features established by the taxonomy from demonstration videos. We later integrate the aforementioned motion feature extractor into the state-of-the-art action recognition model and show that this combination outperforms the baseline accuracy. We also introduce an action recognition framework that learns to embed the visual features and narrated text into a shared embedding space, eliminating the necessity to use one-hot vectors.

In *Chapter 2*, we introduce the *motion taxonomy* and how we use it to embed the motions into *motion codes*, a low-dimensional vector that represents the motions in the human-readable way. We describe the architecture and the learning process of the multiple iterations of the motion prediction model. Finally, we incorporate our motion prediction model into state-of-the-art action recognition model and compare their performances.

In *Chapter 3*, we introduce our framework for learning action recognition model via cross-modal embedding techniques. The model represents the labels via word-level embeddings and eliminates the shortcomings of the one-hot vector encoding. We compare our model to the traditional state-of-the-art action recognition models and provide some discussion on benefits of this strategy.

1.3 Prior Work

1.3.1 Embedding in Deep Learning

The application of embedding concepts became quite popular in recent years. One of the most common issues in machine learning is figuring out how to properly wrangle large amounts of unstructured data when creating models. Embedding data into a lower dimensional space seems to aid greatly in alleviating this issue. Embedding can be described as the translation of high dimensional data into a lower dimensional form. Some, but not all applications of embedding include natural language processing, image and video captioning, recommendation system building, semantic search, and social network analysis.

In this section, we will describe the prior work that uses the embedding concepts for the natural language processing, image classification, and multi-modal information retrieval tasks.

1.3.1.1 Word-Level Embedding

Natural language processing was among the first areas of machine learning that started utilizing the embedding concepts. The well-known word-level embedding models, Word2Vec [1] and GloVe [2], became fundamental pieces of many modern deep learning frameworks, not only for the Language Models.

Word2Vec model has 2 different approaches. The first one is called the Continuous Bag of Words (CBoW) [27]. The model takes a sentence, removes the word in the middle, and uses a neural network to predict the missing word. The second approach, namely skip-gram [1], works in the opposite direction. We word is fed to the neural network to predict the N words around it. The model uses the output of a single layer as the input to N classifiers that predict each word. All words are represented as one-hot vectors. The idea of both methods is to learn a continuous and

low-dimensional representation for words that were learned based on different contexts, where they were used. The final representation is the output of the penultimate layer of the neural network.

Unlike Word2Vec, GloVe, or Global Vectors for Word Representation [2], consider the global statistics within the entire text corpus in addition to the local statistics of the word neighborhoods. In order to do that, they build the co-occurrence matrix, where each entry represents the number of times the word at column j occurs right next to the word at row i. They use this statistics to put additional constraints in the loss function when learning word-level embeddings. Specifically, they encourage the model to learn word embeddings, such that the dot product of any two words is proportional to their co-occurrence value.

One of the newest methods for word embedding was introduced in [28] where the model encodes the words on a character-based level. The authors represent each character as a 24-dimensional vector and a single word is represented as a matrix. This matrix is passed through 2 fully connected layers and returned as a word-level embedding, which is ready to be consumed by text encoding recurrent neural networks, such Long Short Term Memory (LSTM) [29] or Gated Recurrent Units (GRU) [30]. This kind of word-level embedding does not require extra space for thousands of word vectors as it depends on the number of letters in the alphabet and the size of fully connected layers.

Another word embedding model, Embeddings from Language Models (ELMo) [3], feeds Word2Vec representations for each word in the sentence to a bi-directional LSTM that adjusts the Word2Vec representation to the local context within the given sentence. This is one example of using the words embedded via Word2Vec in another framework. Similarly, the current state-of-the-art language models, such as transformers [31] and Bidirectional Encoder Representations from Transformers (BERT) [32], also use the Word2Vec embeddings as the inputs to their models.

1.3.1.2 Attribute-Based Embedding

Another application of embedding can be found in some image classification frameworks. For instance, in [6, 7], Akata *et al.* introduced an attribute-based label embedding to categorize images with the classes that share common features (e.g., animal classes). They annotated images with binary codes that represent the existence or absence of the corresponding attributes in the image. The final continuous embedding for each class is computed by averaging the binary vectors in each class. The learning algorithm then uses the new representation for labels and maps the image features to the same vector space. It learns the image encoding function that will minimize the distance between the image vector and the correct label embedding.

Their work mostly followed the idea of using common attributes for different classes, which was previously introduced by Lampert *et al.* [8, 9]. They call their framework Direct Attribute Prediction (DAP), which feeds the input image to the binary classifiers for each attribute to identify the probability of existence of each attribute. They further use these attribute probabilities to find the class with the highest probability.

1.3.1.3 Cross-Modal Embedding

Cross-modal embedding is a technique mostly applied in information retrieval frameworks, where the query and the retrieved data have different modalities. A common example is retrieving text data from visual data and vice versa [33, 34]. Many deep learning frameworks use this technique for other methods besides cross-modal information retrieval. For instance, Wehrmann *et al.* [28] and Laina *et al.* [35] use cross-modal embedding to caption images. Similarly, [36, 37, 38] incorporate this method for video captioning and video action retrieval. Our work in Chapter 3 was mostly inspired by the framework from [36]. Wray *et al*. use cross-modal embedding for the

fine-grained action retrieval via aligning visual data to each Part of Speech separately, which is quite similar to the action recognition models.

All mentioned frameworks embed the visual data and semantic data into a shared embedding space. They mostly use pairwise ranking loss function to align the matching video-text pairs, while pushing away irrelevant pairs further. The pairwise ranking loss, also known as triplet loss, takes one vector from one modality and two vectors from other modality. One of these 2 vectors must be positive and other must be negative match to the former vector. It computes the similarity scores or distances between the single vector and other 2 vectors. The goal is to make the similarity score between the positive pair higher than the negative pair, and vice versa if using distance between vectors instead.

1.3.2 Action Recognition

The success of convolutional neural networks in image classification by training on large-scale ImageNet dataset [39, 40] let the researchers extend the image classification models to play a fundamental role in video action recognition domain. Currently, there are numerous deep learning models designed for the purpose of action recognition that use convolutional and recurrent neural networks [41, 18, 23, 19, 42, 21, 43, 20, 44, 22, 24, 45]. These methods achieved state-of-the-art results on well-known datasets, such as UCF101 [46], HMDB51 [47], and EPIC Kitchens [48].

The work by [41] first came up with idea to use ConvNets for video classification. They studied different ways of combining the spatial information from video frames, introducing early fusion, late fusion, and slow fusion strategies. The early fusion strategy extends the first convolutional layers by adding a temporal dimension. This method allows to calculate the motion features immediately using the pixel-level information. The late fusion takes the first and the last frame of a short clip and averages the outputs of their final layers. This is somewhat similar to

what the modern two-stream architectures do to combine the outputs of two modalities. The slow fusion method uses multiple parallel convolutional layers with early fusion, which are gradually combined in the higher layers via late fusion.

After their work, [18, 19, 20, 21] used a two-stream approach, where two separate ConvNets are trained on two input modalities, namely an RGB frame and stacked optical flow vectors. In all cases, the outputs of two models were late fused, with exception of the model from [20] that fused the modalities at the early convolutional layers, since the authors believed that the feature maps of both modalities must be combined at the early stage to associate separate spatial regions. After the publication of [18], it became clear that optical flow features are quite beneficial for the action recognition since they represent the motion in the video. Nowadays, it is rare to see action recognition frameworks that do not utilize optical flow frames at all. The later works, such as [19], use the idea of feeding a single RGB frame to one model to get spatial information about objects and the stacked flow frames to the other model that calculates the motion features. However, [19] uses much deeper CNN models, such as Inception [49] and VGGNet [50], and pre-train the model with ImageNet, which let them achieve performance comparable with the accuracy of image classification. A year later, [21] introduced Temporal Segment Networks (TSN). The model segments the input video into N segments and randomly samples one RGB frame from each segment. As previously, they stack the optical flow frames, but now they do this for each segment as well. At the end, they combine the results by averaging, since this method produces the better results than max pooling and weighted averaging. The authors also use two new modalities, namely the difference between RGB frames and warped optical flow frames. They compute the latter modality by estimating the homography matrix to compensate the camera motion.

One shortcoming of the aforementioned methods is that they do not properly leverage temporal information of the videos from RGB frames and mostly delegate this task to optical

flow frames. This issue also limits their capability to classify longer demonstrations of action; only the TSN model, proposed in [21], was able to handle that via a sparse temporal sampling method. Another solution to this problem was to use recurrent neural networks to preserve temporal ordering of the frames, which was done in [42, 22, 44] with LSTM and bidirectional LSTM layers. An alternative solution was presented in [43], where authors summarized the entire video sequences into a singe image and used conventional image classification models.

1.3.2.1 Inflated 3D ConvNets

Other methods of video action recognition are based on the idea of adding a temporal dimension to the ConvNets. The first successful implementation of this method was done in [23], where they show that 3D convolutional layers are more suitable for preserving temporal information. The later work in [24] introduces two-stream inflated 3D ConvNets (I3D) that demonstrates significant improvements in action recognition in UCF101 and HMDB datasets. The main contribution of this work was the idea of inflating 2D image classification networks with temporal dimension and pre-training it on a new large-scale Kinetics dataset, which contains 400 activity categories with over 400 videos in each category.

Throughout our research, we use the I3D model in our experiments. We use it as the base model to extract visual features from the videos, as well as the baseline to compare with. We decided to use this model since it achieves the current state-of-the-art results on the EPIC Kitchens [48] dataset that we use in our experiments. Nonetheless, we applied some limitations to this model to make the training process faster and to allow our model use larger batches. Specifically, instead of the recommended 64 frames, we only sample 12 frames per video. Otherwise, the model would need much more computational power. For instance, the authors of [24] use 64 Nvidia Tesla K40 GPUs to train this model with larger batches.

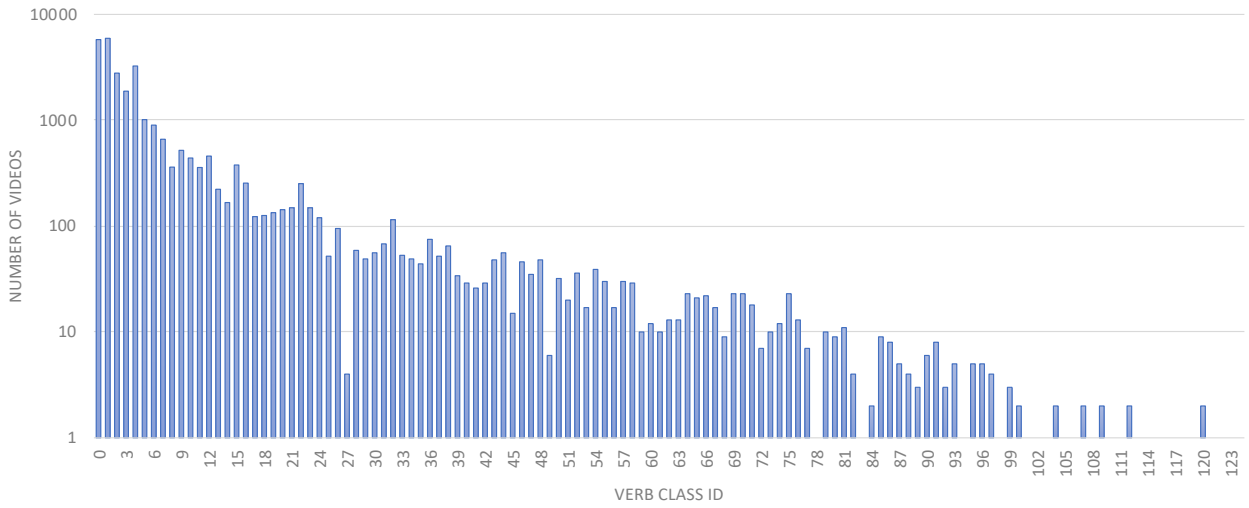


Figure 1.1: Distribution of verb classes in EPIC Kitchens dataset. The number of videos are shown in logarithmic scale.

1.3.3 EPIC Kitchens Dataset

EPIC Kitchens [48] is a large-scale egocentric dataset of cooking videos. It has 28K action segments from 32 different participants. All videos were shot at 60 frames per second and one segment on average is 4 seconds long. The dataset also includes optical flow frames with 30 frames per second that were extracted with TV-L_1 algorithm [51]. We use this dataset throughout our experiments. Each video segment in the dataset was annotated with open narrated sentences, as well as with separate nouns and verbs. The verbs were grouped into 125 classes, while nouns were grouped into 331 classes. Authors also provide the bounding boxes for the objects for object detection and recognition tasks. In our research, we are mostly interested in the verb recognition sub-field of action recognition. Figure 1.1 shows the distribution of verb classes in the dataset in logarithmic scale. We can see that the classes are quite unbalanced, with only six classes that have at least 1,000 videos, namely *take*, *put*, *open*, *close*, *wash*, and *cut*. Moreover, these classes compose more than 70% of the entire dataset. Out of all 125 verb classes, only 26 of them have at least 100 videos.

Chapter 2: Motion Embedding with Motion Taxonomy

In this chapter, we introduce the *motion taxonomy* and its application in *motion embedding*. First, we define the motion taxonomy and the rationale behind its development. Then, we describe our methods for using the taxonomy to obtain low-dimensional motion representations and learning a deep model for predicting motions from the demonstration videos. Lastly, we introduce our action recognition framework that incorporates the extracted motion features in the learning and prediction process.

2.1 Motion Taxonomy

Most of the current action recognition datasets contain videos that were annotated with narrated sentences. The sentences are parsed into parts of speech, such as verbs and nouns, which are later grouped into classes. Some datasets also include the bounding boxes for object detection frameworks. These sentences in general provide sufficient semantic representations for the actions in the videos, especially if the videos demonstrate activities for specific domains, such as cooking. Nonetheless, it can be difficult to precisely visualize the actions by only looking at these annotations. This is because actions can be ambiguous from mechanical perspective and semantically identical actions can be executed differently. For instance, one person could flip an omelette using a spatula, while other person could do it by tossing the pan. The mechanical aspects of the actions depend on the motions that the actor executes. In order to achieve good motion representation, we developed a *motion taxonomy* [25, 26].

Motion taxonomy is a hierarchical tree-like structure that defines the motion components. The components include interaction type between the actor and the object, their individual trajectories, how many hands are involved in manipulation, and whether the actor uses any tool for manipulation (e.g., knife, spatula, etc.). The components can be correlated by some degree, but for the following experiments, we assume they are all independent from each other.

By following the branches with certain decisions and concatenating the outcomes from each tree, we derive a low-dimensional binary vector that represents the given motion. We refer to such representation as the *motion code* and to its derivation process as the *motion embedding*. The motion code encapsulates the most vivid features of one atomic action between the active and passive objects. Thus, the motion code eliminates the mechanical ambiguity of the given actions. Figure 2.1 demonstrates the original version of the taxonomy.

2.1.1 Describing the Interaction Type

At the highest level, the manipulations can be classified as *contact* and *non-contact*. The most obvious example of a non-contact interaction is pouring. There is no direct contact between the receiving container and the source container. In this scenario, one could argue that there is a direct contact between the hand and the source container, but our framework considers the union of these two objects as a single active object in this action. However, this assumption can be false in the atomic action that follows the pouring. For instance, after pouring an oil, the next action could be placing the oil back to its original location. In this action, the hand is the active object, while the oil becomes the passive object. When deriving the motion codes from the taxonomy, it is important to first identify the active and passive objects before proceeding further.

The *contact* motions can be further branched into *engagement type* and *contact duration*. The contact is considered continuous if the active and passive objects are in contact with each other

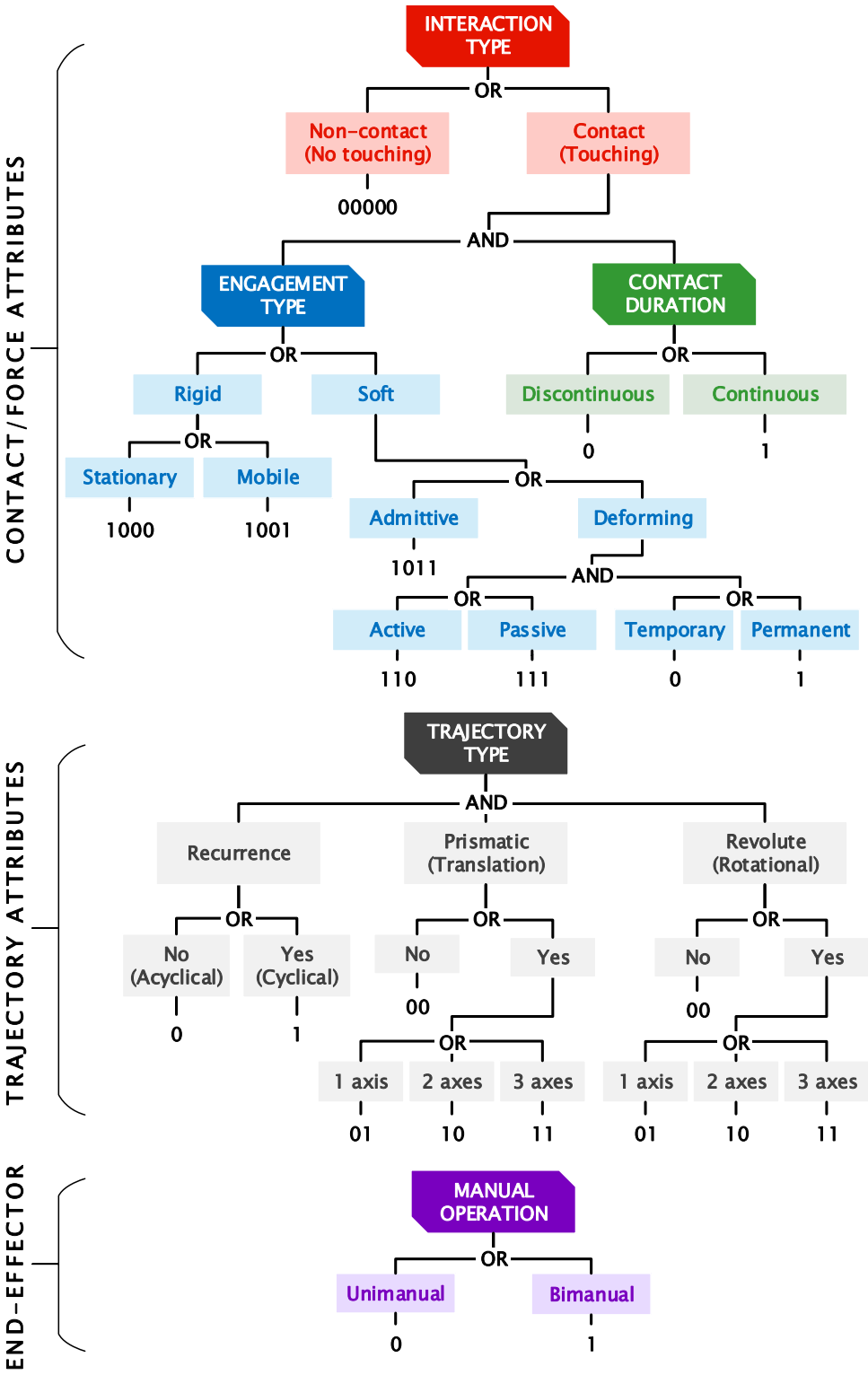


Figure 2.1: The original motion taxonomy. We derive the motion code by concatenating the outputs of each tree.

for most of action duration and if their contact remains persistent throughout that duration. For instance, cutting a potato should be considered as discontinuous if the actor interrupts the contact between the knife and potato each time when he raises the knife. On the other side, if the actor performs the same action without interrupting the contact, it can be considered as continuous. The given example also demonstrates that motion taxonomy can distinguish the differences in performing the same action in multiple ways.

The contact can be also *rigid* or *soft*. The rigid contact does not affect the structure of any object and the only variable component of such interaction is whether the passive object changes its location as the result of interaction. A general example of such interaction is picking and placing the object. The soft contact occurs either when there is admittance between two objects (e.g., dip a meat into sauce) or when one object changes its shapes (i.e., deform). When deforming, either active object or passive object changes its shape temporarily or permanently. The actions that involve cutting, mixing, squeezing, and other motions that change the shape of the objects are considered soft.

2.1.2 Describing the Motion Trajectory

We define the trajectory based on 3 components, namely *recurrence*, *prismatic motion*, and *revolute motion*. During the motion embedding of an action, we identify all 3 components for both the active and the passive objects.

For the prismatic (or translational) trajectory, we need to identify whether there is any translational motion and how many degrees of freedom (DOF) it has. Prismatic motion can have 1 DOF (along a single axis), 2 DOF (along a 2D plane), and 3 DOF (confined to the entire space). Since we are dealing with ego-centric videos, we do not define the axes and the coordinate system. For instance, we consider the prismatic motion to have 1 DOF if its trajectory would draw a straight

line, even though the same video may have another atomic action that was labeled as prismatic with 1 DOF, but the planes of their trajectories are not parallel to each other. However, in the well-defined environment with a coordinate system and visual data from different angles would make it possible to define the trajectories with respect to the world axes.

The revolute (or rotational) motions also can have up to 3 DOF. However, in the kitchen scenario, the most frequent occurrence of the revolute motion will have one degree of freedom. A typical example is opening or closing the door. The door has a single rotational axis and opening or closing it makes the door to rotate around that axis. Another examples are flipping and pouring.

The prismatic and revolute motions are not mutually exclusive. More than that, in most cases of revolute motions, there will also be a prismatic motion. It is a non-trivial and non-natural task for a human to perform actions without any prismatic motion. For instance, during pouring, it is common to raise the hand along with rotating the wrist. The rotational motions with no translation can be achieved when the object is fixed and its motions are constrained to rotate along the axis (e.g., opening the door).

In addition to prismatic and revolute motions, the action can incorporate a recursion. It is quite common to see recurrence in the motions of activities in daily living actions (e.g., mixing, screwing, dicing).

2.1.3 Manual Operation and Active Descriptor

The last two motion attributes in the taxonomy are *manual operation* and *active descriptor*. The former identifies whether the action involves one hand or both hands. The operation is considered to be bi-manual if both hands are executing visually identical tasks. For instance, the most common case of washing a dish will involve both hands holding the dish and wiping it with a sponge. One

the other side, if the action is cutting and one hand is gripping the passive object, while the other hand is cutting it with a knife, these operation will be labeled as uni-manual.

The *active descriptor* attribute is the most recent addition to the taxonomy. Hence, it was not demonstrated in Figure 2.1. We use it to identify whether the actor is using any kind of tool or utensil in order to execute the action. The tools include a utensil when mixing or flipping, a sponge or towel when washing, a bottle or container when pouring, a knife when cutting, etc.

2.1.4 Taxonomy Revision

During our revision, we identified on flaw in *interaction type* tree of the taxonomy from Figure 2.1. The structural outcome of the objects does not necessarily represent the motion. Change of the objects' shapes or locations is more relevant to the state recognition and transition domain. In addition to that, the object state can be changed even if there is no direct contact between active and passive objects. For example, after pouring, the state of both objects are changed. Therefore, we decided to exclude these attributes from the *interaction type* tree. We still keep them in the taxonomy as a separate tree. However, we rarely use it in our experiments. The last modification was elimination of the bit that represents the number of hands involved. This attribute was not consistent for many actions, since the second hand could appear in the scene, but is not actually involved in the action. This inconsistency could bring confusion during the annotation, as well as during the motion prediction from the videos. Figure 2.2 shows the current version of the taxonomy.

2.1.5 Data Annotation

We annotated EPIC Kitchens dataset [48] with the motion codes by watching the videos and following the taxonomy. Throughout our research, we had 3 annotation stages. At the

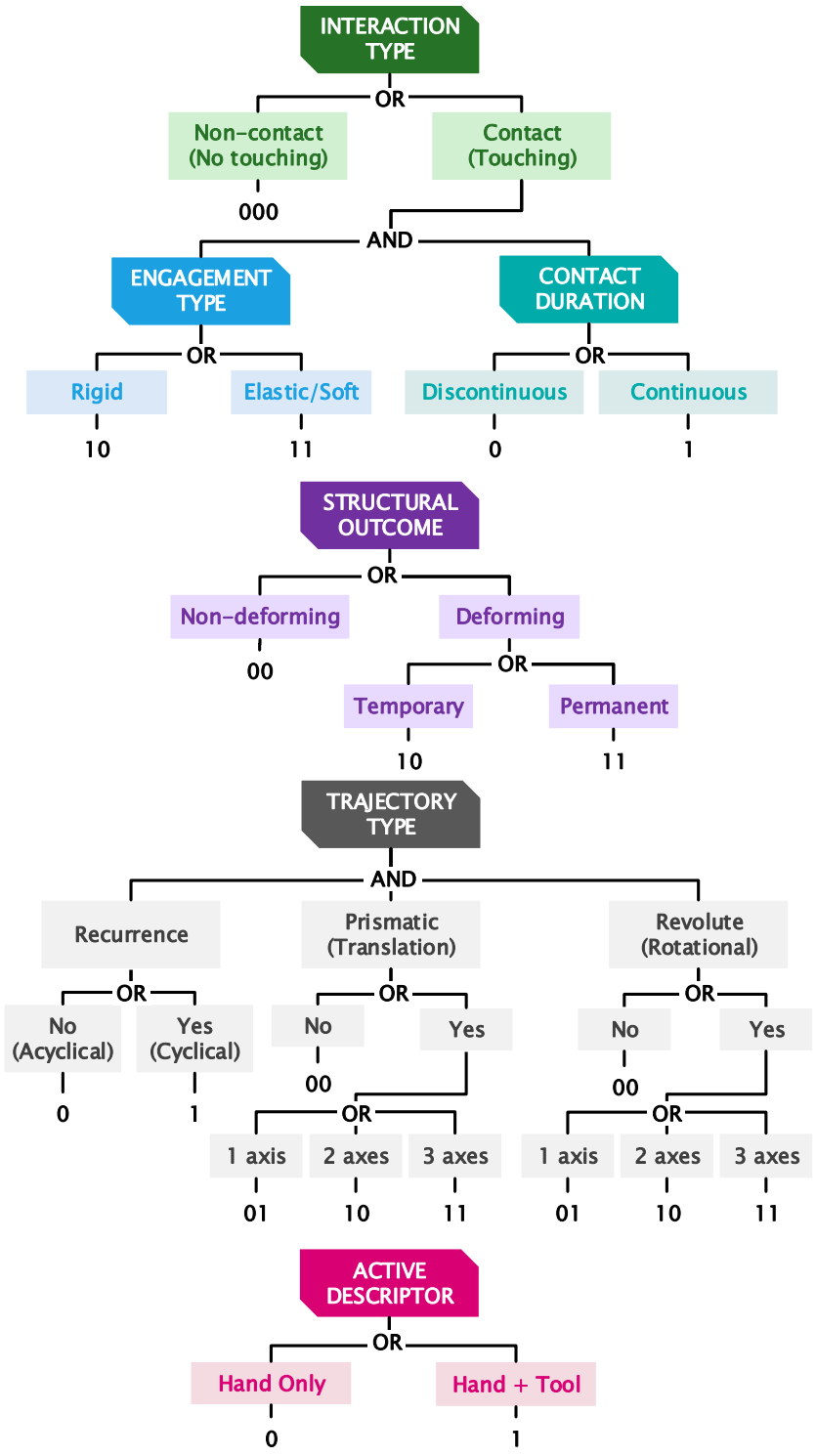


Figure 2.2: Revised motion taxonomy. The structural outcome and trajectory type are derived for active and passive objects separately.

Table 2.1: Dataset after the *initial annotation*. One verb class has exactly one motion code.

Verb	***Motion code***	***Description***
Take	10011-010-0	Rigid, mobile, and continuous contact; only prismatic trajectory; small passive object.
Put	10011-010-0	Rigid, mobile, and continuous contact; only prismatic trajectory; small passive object.
Open	10011-010-1	Rigid, mobile, and continuous contact; only prismatic trajectory; large passive object.
Close	10011-010-1	Rigid, mobile, and continuous contact; only prismatic trajectory; large passive object.
Wash	10001-101-0	Rigid, stationary, and continuous contact; recurrent and revolute trajectory; small passive object.
Cut	11110-110-0	Permanently deforming passive object, discontinuous contact; recurrent and prismatic trajectory; small passive object.
Mix	10111-111-0	Admittive and continuous contact; recurrent, prismatic, and revolute trajectories; small passive object.
Pour	00000-001-0	No contact with passive object; revolute trajectory; small passive object.
Shake	10011-111-0	Rigid, mobile, and continuous contact; recurrent, prismatic, and revolute trajectories; small passive object.
Squeeze	11101-010-0	Temporarily deforming passive object, continuous contact; prismatic trajectory; small passive object.

first stage, namely *initial annotation*, we used the original taxonomy. After that, we applied the revisions mentioned in Section 2.1.4 and moved to the second stage of annotation, namely *second annotation*. Finally, we developed a method for annotating the videos automatically, namely *automatic annotation*. We will discuss the last stage separately in Section 2.4.

During the *initial annotation*, we wanted to evaluate the robustness of the defined attributes in the taxonomy. It is important to note that we used the initial version of the taxonomy during this annotation. At that point of time, we did not consider the trajectory of the passive object. This is also the reason why the original taxonomy has the *mobile* and *stationary* attributes in its *interaction type* tree. We selected 10 classes in the EPIC Kitchens dataset. From each class, we selected 100 videos that have visually identical motions (i.e., have the same motion codes). During this annotation, we already eliminated the bit that represented number of hands. However, we

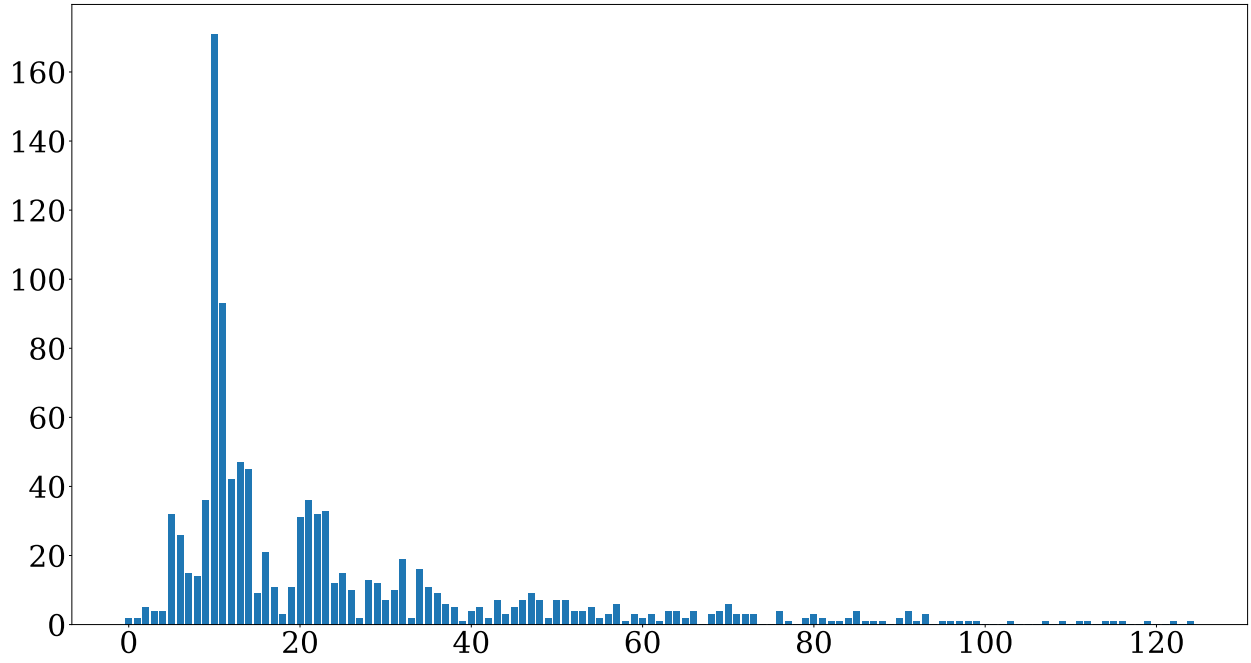


Figure 2.3: Number of unique motion codes per verb class after the *second annotation*.

also temporarily added a new bit that represents the size of the passive object. We did it because 40% of videos would have had identical motion codes otherwise (i.e., *take*, *put*, *open*, *close*). Another modification, or rather a simplification, was reducing the prismatic and revolute trajectory bits to a single bit, which represents whether there is any motion in these trajectories or not. Table 2.1 shows the verb classes and their motion codes.

During the *second annotation*, we randomly selected at most 500 videos from each class. We used the revised version of taxonomy from Figure 2.2. The resulting dataset has 4,488 videos annotated with motion codes. Figure 2.4 shows the distribution of the videos in the dataset with respect to the verb classes they belong to. Figure 2.3 shows how many unique motions we observed in each verb class. We can see that some classes have more than 50 different motion codes (*dry* and *remove*), while 13 classes have at least 20 different motion codes. We should also mention a single modification that we applied during this annotation. Since we are dealing with ego-centric videos, it is almost impossible to precisely identify whether the prismatic motion has 2 or 3 DOF, as well

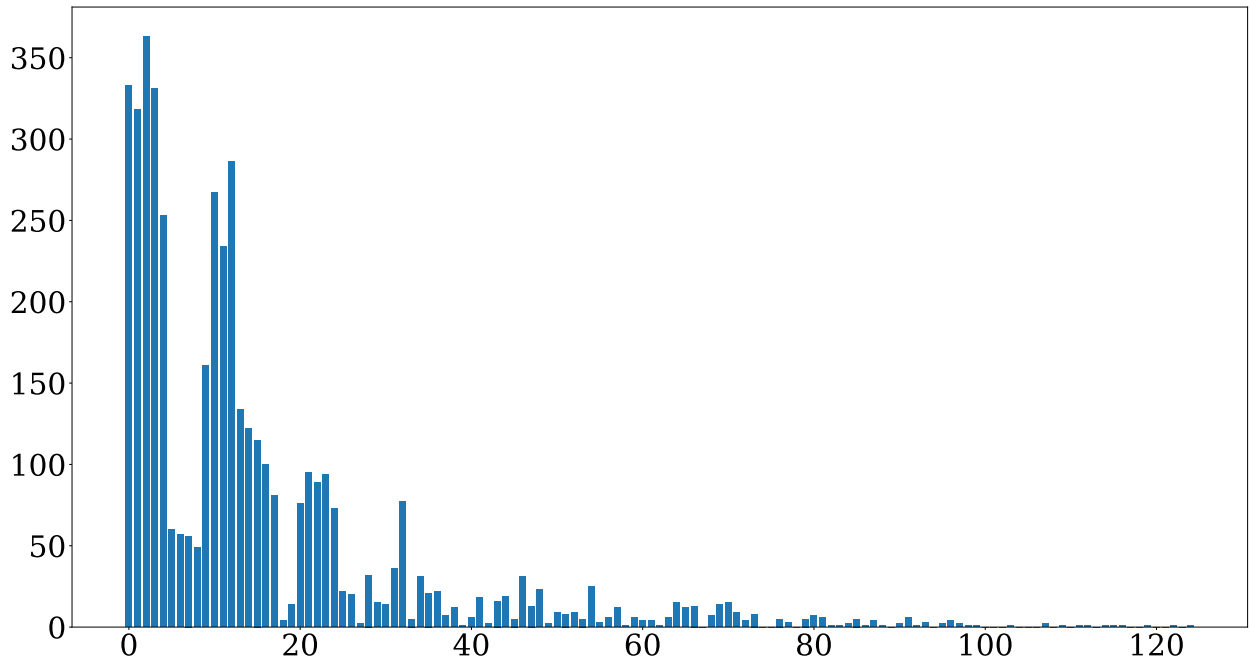


Figure 2.4: Number of videos per verb class after the *second annotation*.

as to identify whether rotational motion has 2 or 3 DOF. Therefore, we simplified the trajectory definitions for prismatic and revolute motions to have 0, 1, or many DOF.

2.2 Motion Prediction

After annotating videos with the motion codes, we started the development of the motion code estimation model. As we mentioned in Section 1.3.2.1, we use the Inflated 3D ConvNets [24] model to extract visual features from the videos. Since the motion codes are represented as binary vectors, one could suggest to use separate binary classifiers for each bit. However, this approach would not fit for the motion code prediction, since some bits depend from each other. For instance, the three bits of *interaction type* attribute depend from each other starting right at the first bit. If it is zero, the remaining bits must be also zero. Therefore, we developed other more suitable methods that we describe in the following subsections.

2.2.1 Learning with a Tree-Like Structure

The motion taxonomy has a tree-like structure. Hence, it would make sense to use this structure in the learning algorithm. In addition to that, at that time, we used the taxonomy from Figure 2.1, which has a relatively deep tree for the *interaction type*. Figure 2.5 illustrates the structure of the model that we developed. We use I3D to extract visual features v_1 and use these features do decide whether there is a contact or not. We do that by feeding features v_1 to a single layer binary classifier. We further feed the same features v_1 to other fully connected layers that are responsible for different tasks. For instance, v_{2a} learns features that can represent the contact duration and we feed them to another binary classifier that identifies whether the contact is continuous or not. Similarly, v_{2b} learns the features to represent the interaction type and these features are used to identify whether the interaction is soft or not (i.e., rigid). As with features v_1, the output of v_{2b} is further fed to 2 neural networks that predict the interaction type attributes, namely v_{3a} that predicts whether the interaction is mobile or not (i.e., stationary) in case of rigid contact, and v_{3b} that predicts whether the contact is deforming or not (i.e., admittive) in case of soft contact. The last interaction attribute is the outcome of the deforming contact. Since these outcomes occur during the deforming contact, the output of v_{3b} is used to calculate the features for the outcome classification. From Figure 2.5 we can also see that there is no branch to identify whether an active or passive objects were deformed during the deforming contact. We did not include this branch in order to reduce the complexity and confusion as there are many cases when both active and passive objects are deformed (e.g., squeezing a sponge).

We can see that this model preserves the tree-like structure of the motion taxonomy. The model encourages the feature vector v_1 to properly represent all interaction features in the video because all lower-level interaction attributes depend on it. Similarly, the layer v_{2b} must learn a

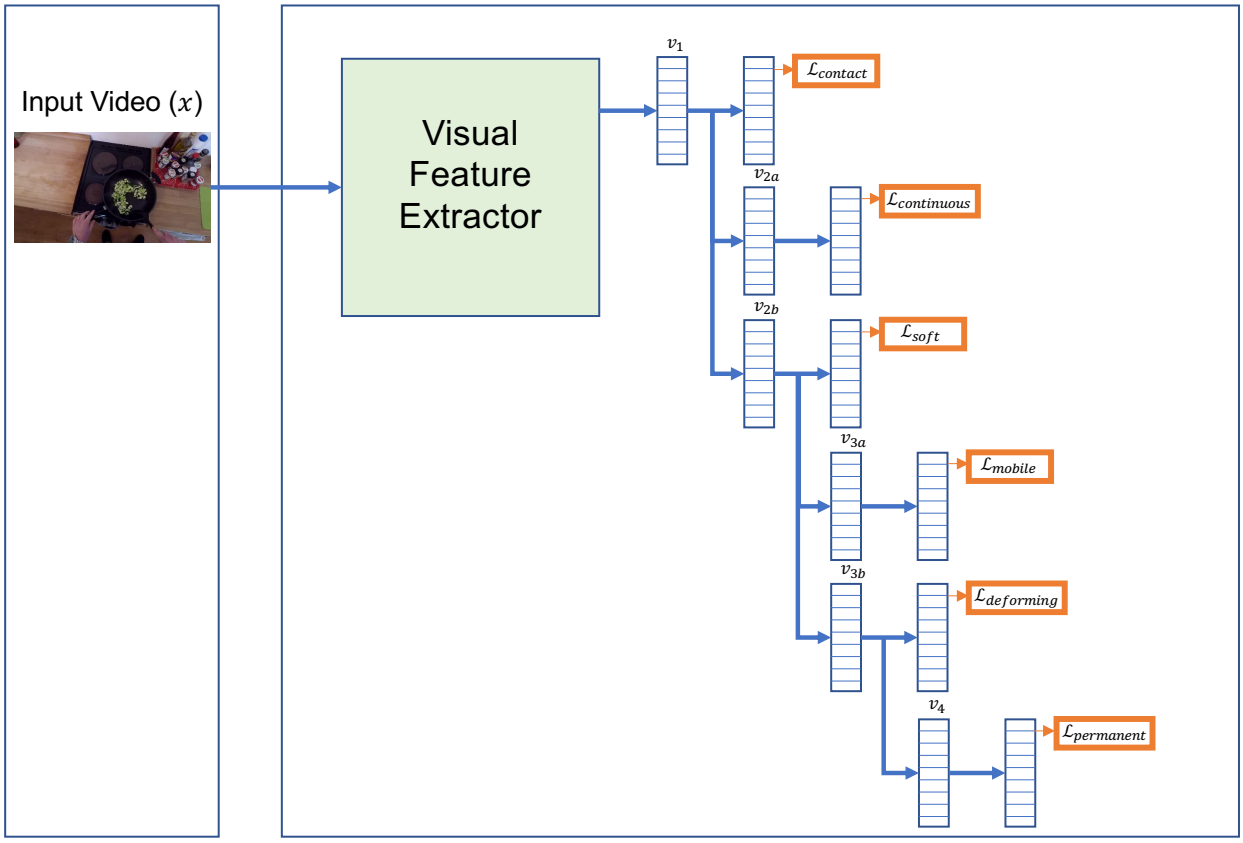


Figure 2.5: Decision tree-like model structure for interaction prediction. The model follows the structure of the *original taxonomy* from Figure 2.1.

representation that can be sufficient for all interaction type attributes in addition to identifying whether the contact is soft or rigid.

Since we want to classify each attribute, all loss functions are categorical cross-entropy functions. Even though we need to classify each attribute, some branches must be ignored for particular videos because the video does not demonstrate these attributes. For instance, if the interaction is non-contact, then the model should not adjust the layers after v_1, except the contact classifier. Similarly, if the contact is rigid, the layers below v_{3a} are not modified. To achieve this behavior, we multiply the loss values by a mask, where the entry is zero if the corresponding attribute is not in the video.

For the remaining bits, we use binary classifiers that also obtain the latent visual features from the deep 3D CNN. This results in 5 separate models for RGB frames and 5 models for flow frames. We consider recurrence, prismatic trajectory, revolute trajectory, and passive object size bits as independent from each other and from the interaction type. As the interaction type model, they train with categorical cross-entropy loss function.

The model was trained with the dataset from the *initial annotation*. We split the dataset to train and validation sets via 80/20 split. The model was trained for 100 epochs with Adam optimizer and initial learning rate of 10^{-4} that was reduced to 10^{-5} after 60 epochs.

2.2.2 Learning to Classify Motion Components

The model with a tree-like structure perfectly aligns with the hierarchical nature of the taxonomy. However, that model has three drawbacks. First is its computational complexity. The model learns multiple representations for each attribute and it derives them from the higher-level attribute representations vial fully connected layers. In addition to that, it requires separate deep 3D CNN models for each tree. In total, this framework employs 10 separate deep models, 5 for RGB frames and 5 for optical flow frames. Second, we also want the model to learn a continuous embedding that we can later use as a high-dimensional motion embedding. Finally, the revised version of the motion taxonomy from Figure 2.2 has more shallow trees than the original and the tree-like model structure will not necessarily benefit the performance.

We designed a new learning method after considering the mentioned drawbacks. This time, there is only one I3D model per modality that extracts a high-dimensional visual feature vector. We further use the same visual features to classify each independent motion component. As we mentioned in Section 2.1, we assume that all trees in the taxonomy are independent from each other. There can be small correlations between each tree, but we consider them to be weak

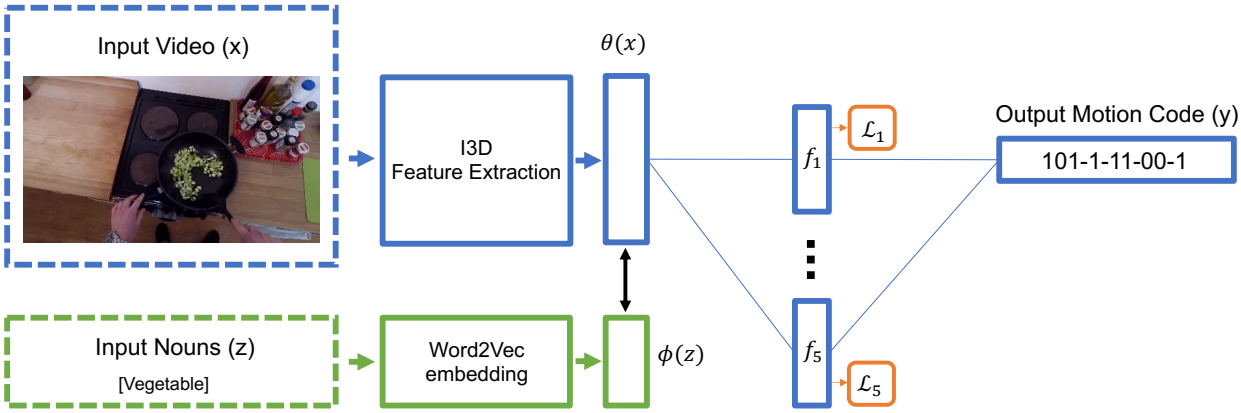


Figure 2.6: Second motion prediction model. The model classifies the independent components of the *revised motion taxonomy* from Figure 2.2. If the semantic data $\phi(z)$ is used, the vector $[\theta(x), \phi(z)]$ is fed to another fully connected layer.

enough to define them independently from each other. Additionally, we consider the subtrees of the trajectory type to be independent as well, since any combination of prismatic and revolute trajectories is possible, while recurrence has only one special scenario when it cannot be 1 (i.e., when there is no motion at all).

By looking at the taxonomy in Figure 2.2, we can identify the following independent motion components: the interaction type, trajectory recurrence, prismatic trajectory, revolute trajectory, and the active descriptor. We do not consider the structural outcome for motion prediction since it is not within the scope of the motion prediction. The interaction type has 5 possible outcomes, which means the model must classify with one of five classes. Similarly, the recurrence and active descriptor have 2 classes each. The prismatic and revolute trajectories have 3 classes each (i.e., 0, 1, and many DOF).

For this model, we used the dataset from the *second annotation*. We annotated 4,488 videos with the motion codes, which is not sufficient to train a model that classifies all mentioned components. Therefore, we had to further simplify the format of the motion codes for this dataset. More specifically, as with the original taxonomy, we only consider the trajectory of the active object, while the trajectory of the passive object was reduced to a single bit. That single bit represents whether

the passive object is moving with respect to the active object or not. We reduced the trajectory by directly comparing the full-fledged passive object trajectory to the active object trajectory. If the trajectory bits are identical, we assume that the passive object is static with respect to the active object. Otherwise, the passive object has a relative motion. We understand that such a method will produce some noise to our annotations since there can be cases when the passive and active objects have an identical number of DOF for translational and rotational motions, but their directions or coordinate systems are different. The last modification was the removal of the active descriptor because it was less significant for the motion representation in comparison with the remaining components. As a result, we have 5 components: interaction type (5 categories), recursive motion (2 categories), prismatic trajectory (3 categories), revolute trajectory (3 categories), and the passive motion (2 bits). In total, there are 180 possible combinations for this motion code format.

After applying the mentioned adjustments, we balanced the dataset. We removed the videos that have motion codes with less than 20 samples in the dataset. We also removed the videos that belong to the verb class with less than 20 samples in the dataset. The latter filtering was applied because we wanted to integrate the motion prediction model to the action recognition model. We split the resulting 3,528 videos into a training set with 2,742 videos and a validation set with 786 videos.

The model can be defined as follows. We feed the video $x \in \mathcal{X}$ to the deep network $\theta : \mathcal{X} \rightarrow \Omega$. We use the resulting feature vector $\theta(x)$ as the input to 5 classifiers to predict the motion components and concatenate them into the output motion code. The loss function is the linear combination of all categorical cross-entropy loss functions:

$$\mathcal{L} = \sum_{i=1}^{5} \lambda_i \mathcal{L}_i \tag{2.1}$$

To potentially improve motion code prediction, we also tried to incorporate knowledge of the objects in action into the training process. Formally, we modified the model described above by encoding the semantic features of the objects $z \in \mathcal{Z}$ with embedding function $\phi : \mathcal{Z} \rightarrow \Psi$ and combining it with the visual features. For our experiments, we concatenate these two feature vectors. We use a Word2Vec model pre-trained on Google News [1] (containing over 3 million words) to encode these semantic features about objects seen in each video. A model of this kind could be used for queries, where we can determine what kind of motion with a given object can be executed to replicate a certain activity.

The entire model was trained for 50 epochs with Adam optimizer and the learning rate set to 0.0003 that decreases by 40% every 5 epochs. For the first 3 epochs, the convolutional layers of the base model were frozen to allow the top layers to fine-tune for a better initialization. The input video frames are sampled to 6 frames per second to increase the training and inference speed. We use both RGB and optical flow frames. In our experiments, all λ coefficients are set to 1. During training, we add the L_2 norm of the network parameters multiplied by a weight decay factor to the loss function for L_2 regularization [52]. The model was implemented with the TensorFlow library [53]. The overall architecture is illustrated as Figure 2.6.

2.3 Action Recognition with Motion Features

Motion prediction model provides us with the motion features from the videos. Currently, optical flow estimation is considered to be the primary motion feature extraction method. The work from [22] emphasizes the importance of accurate optical flow computation for the action recognition models. The results of all state-of-the-art action recognition models support that observation. It is common to see the frameworks that learn two separate models for RGB frames and optical flow frames. Since the task is a multi-class classification, the final layers of both models

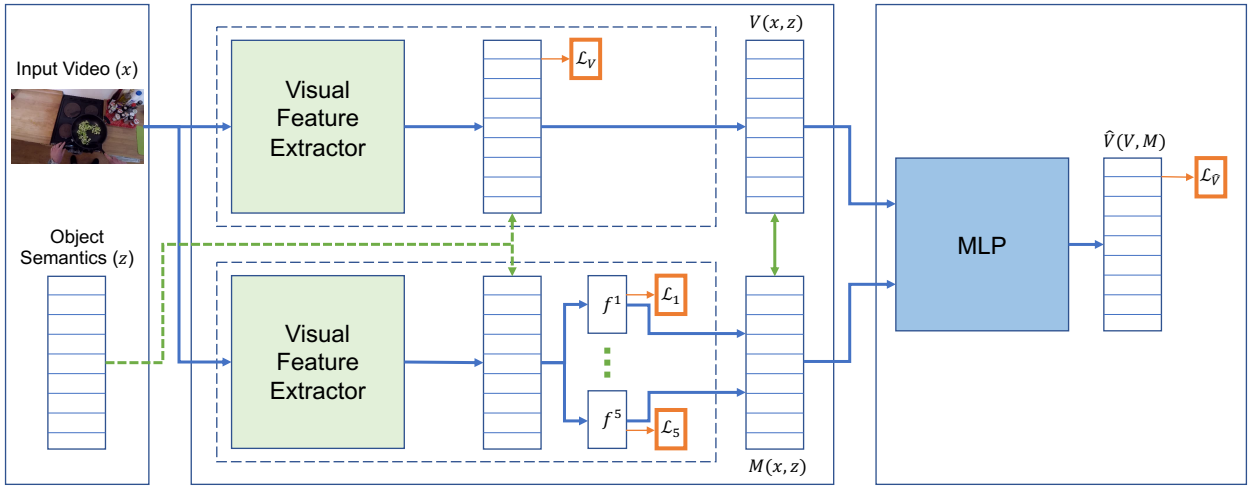


Figure 2.7: The structure of our verb classification framework with motion code predictor. Verb classifier and motion predictor may or may not use the object semantic data.

are N-dimensional vectors, where N is equal to the number of classes in the dataset. Both vectors are averaged to obtain the output of the fused two-stream network, which is further converted into a probability distribution for the classes via the softmax function. In all cases, the accuracy of the fused model is higher than the accuracy of the individual modality models. With that in mind, we wanted to discover how the motion features from our motion prediction model would affect the performance of the action recognition model.

2.3.1 Methodology

Our proposed model can be described as follows. Let $x_i \in \mathcal{X}$ be the i^{th} video in the set of videos and $y_i \in \mathcal{Y}$ be the corresponding verb class label of that video. Conventional classification models first extract the visual features from the video into a latent space as $V : \mathcal{X} \rightarrow \Omega$. Our method augments that model with a motion code embedding model from Section 2.2.2. We concatenate all probability distributions for motion components into a single motion embedding vector. For simplicity, we refer to the entire motion code embedding model as $M : \mathcal{X} \rightarrow \Lambda$. After we get $V(x)$

and $M(x)$, we concatenate them and feed the resulting vector to a multi-layer perceptron with 2 fully connected layers. The output vector $\hat{V}(V, M)$ is the new verb class probability distribution.

Just like with motion prediction model, we were interested to see how the information about the ground truth nouns would affect the performance. As previously, we use the nouns for each video and embed them via Word2Vec pre-trained on Google News [1]. We integrate the nouns into our model in multiple ways. First, we use it in the verb classification model by concatenating it to the output probability distribution and feeding the resulting vector to a fully connected layer that acts as a new probability distribution. We denote this model as $V(x, z)$ Then, we integrate the nouns into motion prediction model by concatenating them to the penultimate layer that is shared by all motion component classifiers. As with verb classifier, we feed the resulting vector to a new fully connected layer, whose output will be fed to the motion component classifiers. We denote this model as $M(x, z)$. We compare different variations of the verb classification model (e.g., with and without predicted motions or nouns) in Section 2.5. Figure 2.7 illustrates the framework structure.

Each verb classification and model prediction model was trained separately for 50 epochs with Adam optimizer and the learning rate set to 0.0003 that decreases by 40% every 5 epochs. For the first 3 epochs, the convolutional layers of the base model were frozen to allow the top layers to fine-tune for a better initialization. The input video frames are sampled to 6 frames per second to increase the training and inference speed. We use both RGB and optical flow frames. During training, we add the L_2 norm of the network parameters multiplied by a weight decay factor to the loss function for L_2 regularization [52]. The models were implemented with the TensorFlow library [53]. The model $\hat{V}$ was trained for 200 epochs with Adam optimizer and the learning rate of 0.0005. We used the same dataset as in the motion prediction model from Section 2.2.2, with 2,742 training videos and 786 validation videos. The dataset has 33 verb classes. Each class has at

least 20 videos in this dataset. For the testing, we sampled 1,517 videos that belong to the 33 verb classes in the training and validation sets.

2.4 Word2Motion Prediction

One obvious flaw of the models and experiments from Section 2.2 and Section 2.3 is the size of the dataset. The *initial* and *second annotations* provide only 1000 and 4,488 videos respectively to be used in training and validation. We acknowledge that this is not sufficient for the models that are based on action recognition frameworks. In addition to that, we had to simplify the motion code format multiple times to be able to achieve acceptable motion prediction performance. The entire EPIC Kitchens dataset contains about 28K videos, which is six times larger than our dataset after the *second annotation*. However, the motion code annotation process is time intensive, so we had to find a way to annotate the dataset with less human effort.

We have previously mentioned that semantic labels do not provide enough information from the motion mechanics perspective. However, we wanted to see what if we learn a model that uses the semantic labels to annotate the videos with the motion codes. We show in Section 2.5 that the integration of nouns into motion prediction model significantly boosts its accuracy. Therefore, we were interested to test how the full semantic labels (i.e., verbs and nouns) can be translated to the motion codes. We understand that this method will provide us with noisy annotations because the semantic labels cannot robustly represent the motions and the motion prediction model trained on the resulting dataset will learn to predict those noisy motion codes. However, we wanted to compensate that noise with the scale of the training set. Moreover, the training of this Word2Motion model does not require too much computational power as the input size is relatively small. In the future work, we plan to use both videos and semantic labels to train a model with similar task. Note that this is different from the motion prediction model, since the

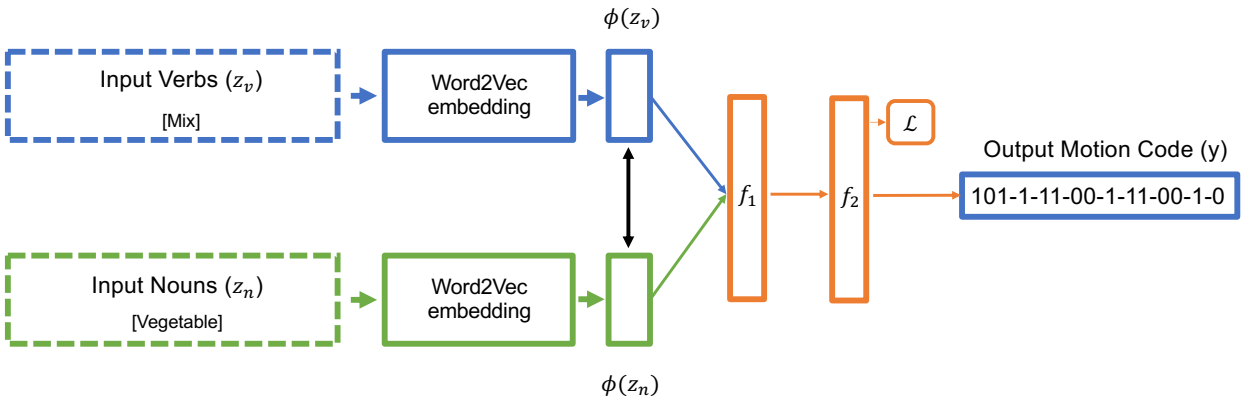


Figure 2.8: The structure of our Word2Motion model. The model takes embedded nouns and verbs and feeds them to a 2-layer MLP. The final layer output is the predicted motion code.

idea of motion prediction model is to only use the video modality with no other information. We previously used the knowledge of nouns for the testing purposes only. In the real world scenario, we assume that visual data is the only information the model will have access to.

2.4.1 Methodology

We implemented the Wrod2Motion model in a relatively trivial way. The future versions will be more sophisticated. Figure 2.8 shows the model structure. We feed the verbs and nouns of each video to the model. We use 100-dimensional Word2Vec model pre-trained on Wikipedia [54, 55, 56] to embed the verbs and nouns separately. The embedded vectors are concatenated into a single input vector. The model has 2 fully connected layers.

We train the model to embed the inputs to the motion codes in two ways. One way is to map the inputs to a vector that represents the actual motion code. The loss function is the squared Euclidean distance between the output vector and the motion code label. We refer to such method as a *bitwise* learning. Another way is to use the approach similar to the motion prediction model from Section 2.2.2. More specifically, use the output of the penultimate layer to classify each motion component. In this case, the loss is the sum of cross-entropy loss functions for each component. We refer to this method as a *componentwise* learning.

Since we want to use this model to automatically annotate the entire EPIC Kitchens dataset, we make the model learn to predict the full-fledged version of the motion codes that follow the format of the taxonomy in Figure 2.2. We use the dataset from the *second annotation* and split it into training and validation sets. We train the model for 150 epochs with Adam optimizer and a constant learning rate of 10^{-4}.

2.5 Results

In this section, we present the results of our models and experiments presented in this chapter. We will first discuss the performance of Word2Motion model from Section 2.4 since we have separate results for motion prediction and verb classification before and after the application of Word2Motion model.

2.5.1 Word2Motion

As we mentioned in Section 2.4.1, we trained the model via *bitwise learning* and *component-wise learning*.

2.5.1.1 Bitwise Learning

Figure 2.9 shows the accuracy trend on the training and validation sets after each epoch. We also include the accuracy values for the motion code prediction with at most 1, 3, and 5 mispredicted bits. In total, there are 17 bits in the motion code. We can see that exact motion code prediction does not get higher than 40%. With at most 1 mispredicted bit, the accuracy is almost 60%, while the accuracy with at most 3 and 5 mispredicted bits reaches 80% and 92% respectively.

Figure 2.10 shows the bitwise accuracy on the validation set before and after training. The values from Figure 2.10a represent what the bitwise accuracy would be if we always select the

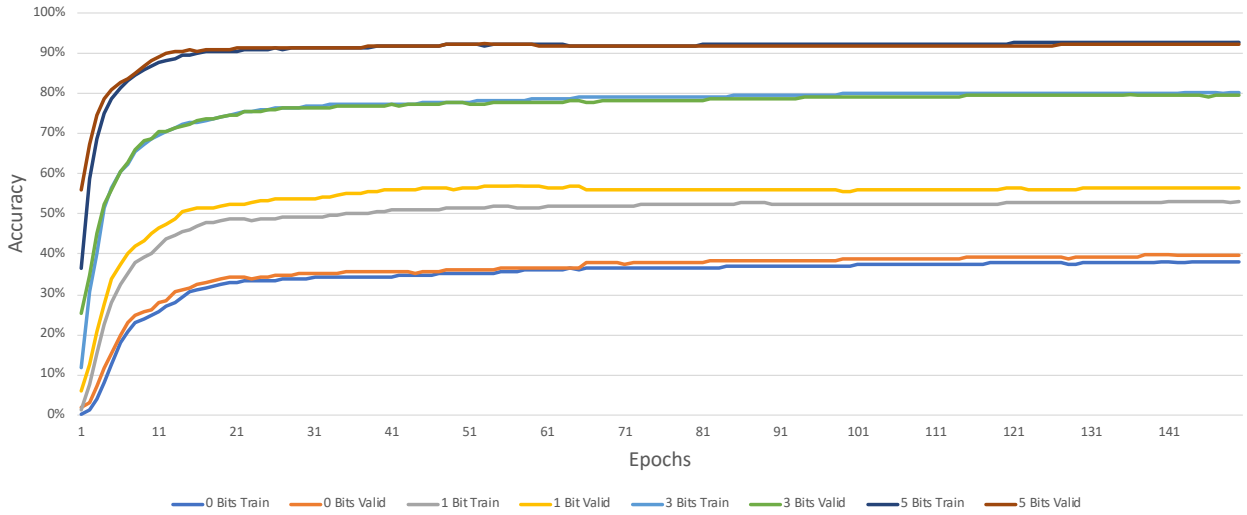


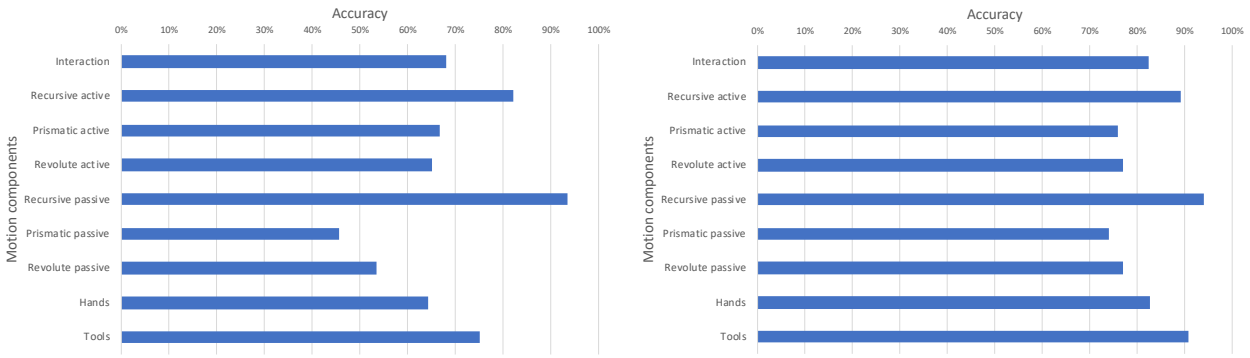
Figure 2.9: The accuracy of the *bitwise* Word2Motion model. The figure shows the accuracy for the exact motion code prediction, as well as the accuracy of predicted motion codes with at most 1, 3, and 5 bits that were wrong.

(a) Before training

(b) After training

Figure 2.10: The *bitwise* accuracy on the validation set before and after training the model.

same value for each bit which is most often found in the validation set. For instance, Figure 2.10a shows that in 80% of samples in the validation set, the bit number 5 is either 0 or 1. We can see that bitwise accuracy increases after training the model with no exceptions. Even the first bit, which in almost 98% of samples has the same value, improves by less than one percent.



(a) Before training

(b) After training

Figure 2.11: The *componentwise* accuracy on the validation set before and after training.

2.5.1.2 *Componentwise Learning*

The *componentwise learning* method achieved similar accuracy performance as the *bitwise learning*, about 40% on the validation set. There are 9 independent components in the motion code: interaction, recursive trajectory, prismatic trajectory, revolute trajectory, number of hands, and whether the actor uses any tool. The trajectory components are separately defined for active and passive objects. As with *bitwise learning*, we provide the accuracy before and after training the model in Figure 2.11. We can see that the accuracy of all motion components imporves after training. It is also interesting to see that the motion components are quite unbalanced. The interaction component has 5 possible outcomes, but almost 70% of the videos show rigid and continuous contact, which is not surprising for the kitchen environment. Moreover, about 60% of all videos in the EPIC Kitchens dataset belong the classes *put*, *take*, *open*, *close*, which in most cases have rigid and continuous contact. We can also see that the most unbalanced components are recursive trajectories for both active and passive objects. However, these components have only 2 possible outcomes each, and it is quite rare to see recursive motions in the EPIC Kitchens dataset. Again, the previously mentioned 60% videos do not have any kind of recursion, and the remaining 40% are not necessarily all recursive.

Figure 2.11b shows that the worst component accuracy is about 74%, which we assume is acceptable for the further automatic dataset annotation. We believe this value can be improved with the visual input data on top of the semantic data. Semantic data cannot fully encapsulate the motion representation, while video and optical flow frames can provide valuable information. In the future work, we will incorporate visual data into learning algorithm as well.

We use the *componentwise learning* method for the *automatic annotation* since this method does not produce invalid combination of bits (e.g., *interaction* has 5 valid outcomes, while 3 bits have 8 combinations). We use the model to annotate all 28K videos in the EPIC Kitchens dataset. This annotation allowed us to use our frameworks from Sections 2.2.2 and 2.3 with the full-fledged version of the motion codes. For comparison, the simplified version of the motion codes has only 180 possible outcomes, while the full version can have up to 6.5K valid outcomes.

2.5.2 Motion Prediction

We have two different models for motion prediction, the tree-like model from Section 2.2.1 and the componentwise motion prediction model from Section 2.2.2. The former uses the dataset after the *initial annotation*, while the latter uses the datasets after the *second* and *automatic annotations*.

2.5.2.1 Tree-Like Model

We first report the results of the tree-like model. The dataset has 800 training videos with 80 videos per class and 200 validation videos with 20 videos per class. The model uses the motion codes embedded via the original motion taxonomy from Figure 2.1. The results from Table 2.2 demonstrate the accuracy on the validation set. As we mentioned in Section 2.2.1, the framework utilizes 5 separate deep models per modality. The models for trajectories and passive object size

Table 2.2: Motion prediction accuracy on the validation set after the *initial annotation*. The model was implemented with a tree-like structure, trained on 800 videos, and evaluated on 200 videos.

Model	***RGB***	***Flow***
Interaction bits	61.85	68.04
Motion recurrence bit	83.50	87.11
Prismatic trajectory bit	83.50	83.50
Revolute trajectory bit	85.05	81.44
Passive object size bit	86.30	84.71
Entire code (direct prediction)	38.14	45.87
Entire code (nearest neighbor)	60.82	61.85

are all binary classifiers since we simplified the prismatic and revolute trajectories to be either 0 DOF or any number of DOF. Hence, the learning to classify these attributes should not be too difficult.

In the dataset, 60% of samples were labeled as *not recurrent*, making the improvement after learning to be 23.5% (60% vs. 83.5%). Similarly with the revolute trajectory, which was labeled to be 0 in 60% of the dataset as well. It also gained over 20% improvement in comparison with always predicting the most likely outcome. On the other side, the *prismatic trajectory* and *size* bits did not improve that significantly. Exactly 80% of the videos were labeled to have prismatic motion and the same amount was labeled to have relatively large passive objects, which makes the accuracy boost less than 5%.

The interaction is the most interesting part of this model. We can see from Table 2.2 that the interaction learning algorithm achieves decent performance, given such a small dataset size. Exactly 50% of the videos in the dataset were labeled as rigid, mobile, and continuous contact. However, 12% and 18% boost for RGB and optical flow frames respectively demonstrates that the model learns to distinguish the interaction attributes. We can also see that the optical flow frames provide significantly higher accuracy. This phenomenon will appear in the subsequent results as well.

The last two rows of Table 2.2 show the accuracy of predicting the entire motion code by concatenating the outputs of all 5 models. We show the results of predicting the motion codes directly from the outputs, as well as finding the motion code observed in the dataset with the smallest Euclidean distance to the output. The dataset has 8 unique motion codes for 10 verb classes because the classes *open* and *close*, as well as *put* and *take*, have identical motion codes. We can see that nearest neighbor method has acceptable performance of 60% accuracy. Nonetheless, in the real world scenario, we will not be able to rely on this method as there can be too many motion codes to compare with. In addition, by assuming that the video will have a motion code from the training set, we will put a limitation to the motion prediction model. We could use this method if we had a large-scale dataset with all possible motions, which is not the case for the EPIC Kitchens dataset. Meanwhile, direct prediction method shows significantly worse results than the nearest neighbor method. However, in comparison with the nearest neighbor method with 8 possible outcomes, the direct prediction has 11 possible outcomes for the interaction type, and 2^4 possible outcomes for the remaining 4 bits, which makes in total 176 possible outcomes. Therefore, we cannot say that 38% accuracy for the RGB frames and 46% accuracy for the optical flow frames is an absolutely poor performance. In fact, the results of the model in the following subsection will not be much different, even though the model had almost 4 times more data after *second annotation*. However, we should keep in mind that the *second annotation* also contains much more unique motions. The direct motion code prediction results also show that optical flow frames perform better than RGB frames. This observation demonstrates the impact of interaction prediction to the entire motion code prediction, which is not surprising given that it has almost as many combinations as the remaining attributes all together.

2.5.2.2 *Motion Component Classification*

The motion prediction via motion component classification model was initially trained and evaluated on the *second annotation* dataset with 2,742 training and 786 validation videos. The model predicts the motion codes embedded via the latest version of the motion taxonomy from Figure 2.2. The first version of this model predicts 5 motion components that are similar to the previous tree-like model with one exception. Instead of the size of the passive object, it predicts whether the passive object is in motion with respect to the active object. In addition to that, the prismatic and revolute trajectories have three options (0, 1, and many DOF) instead of two. As the interaction tree became more shallow in the new taxonomy, the number of possible outcomes was reduced from 11 to 5.

As we mentioned in Section 2.2.2, we also use the knowledge of the objects in action represented as 300-dimensional Word2Vec vectors. However, it is unlikely to have the ground truth nouns available in the real scenario. Therefore, we decided to add some noise to the nouns in order to imitate the scenario when the nouns are provided via an object recognition model. More specifically, we randomly select 20% of videos whose ground truth nouns were replaced by any other noun in the dataset.

Table 2.3 shows the performance of the models. We show the results for the entire motion code prediction, as well as the accuracy of predicting each component separately. In addition to that, we provide the accuracy of the motion code prediction with a tolerance for one wrong bit. We do that by computing the difference between the predicted motion code and the label and computing the magnitude of the resulting vector. If the magnitude is not greater than 1 (i.e., at most one bit is wrong), we consider it as accurate.

We can see from Table 2.3 that motion prediction via component classification has similar results as the previous tree-like framework. However, this framework requires only one deep CNN

Table 2.3: Motion prediction accuracy on the validation set after the *second annotation*. The model was implemented with motion component classification.

Models	***RGB***	***Flow***	***Fused***
Baseline			
Entire code	35.1	35.2	38.9
Entire code with 1 bit off	67.3	64.5	70.9
Interaction bits	85.8	84.7	87.0
Recurrence bit	90.7	91.0	92.5
Prismatic trajectory bits	70.6	72.8	73.2
Revolute trajectory bits	74.4	76.2	78.5
Passive motion bit	68.6	64.8	71.9
Nouns			
Entire code	45.3	46.1	48.0
Entire code with 1 bit off	73.2	72.1	75.3
Interaction bits	86.4	86.4	87.9
Recurrence bit	90.6	91.2	92.1
Prismatic trajectory bits	76.0	74.4	74.9
Revolute trajectory bits	80.5	78.6	81.3
Passive motion bit	76.0	78.9	79.9
Nouns (20% noise)			
Entire code	40.8	39.9	43.1
Entire code with 1 bit off	70.7	68.8	72.1
Interaction bits	86.5	85.9	88.0
Recurrence bit	90.7	91.5	92.5
Prismatic trajectory bits	74.2	71.8	73.5
Revolute trajectory bits	76.3	76.6	78.4
Passive motion bit	72.4	72.6	73.9

per input modality instead of 5. Table also shows that interaction prediction is much better than in Table 2.2, which is most likely because the component now has twice less possible outcomes. On the opposite, one additional outcome for prismatic and revolute trajectories resulted in about 10% accuracy decline. Interestingly, the hardest component to estimate is the passive object's motion bit. This is quite surprising due to the fact that it has only 2 outcomes. Nonetheless, the model actually learns how to predict this component as it has relatively uniform distribution in the dataset

(54% 1s and 46% 0s). The table also shows that the late-fusion of the outputs from RGB and optical flow models consistently improves the performance.

Probably the most interesting observation from Table 2.3 would be the impact of the nouns on the accuracy. We can see that ground truth nouns provide about 10% boost over the baseline model for all modalities, which is quite a significant jump. If we look at the breakdown of individual components, noun vectors mostly benefit the prediction of passive object motion with respect to the active object. We observed that majority of videos that were assigned incorrect motion codes by the baseline model but were then given correct codes from the noun model had only 1 bit that was wrong, which was the passive object motion bit. For instance, the baseline model classified a video as passive object motion being present with respect to the manipulator, while the ground truth is the opposite. Interestingly, 80% of these videos showed an action where person either picks up and places an object or opens and closes it (e.g. door, microwave, fridge). The passive object motion bit for pick-and-place actions were corrected from 1 to 0, while in open and close actions, the bit was set from 0 to 1. In almost all cases when a person picks or places an object, the object is moving with the same trajectory as the hand, making the passive object stationary with respect to the hand. On the other side, when person opens or closes a door, the door always moves strictly around its axis, making it to have only a revolute motion with 1 DOF. Such kind of a motion trajectory is rarely performed by a human. These examples show that the model leverages the information about the objects and can make assumptions based on that knowledge. Even while adding 20% noise to the input noun vectors, the overall accuracy of motion code prediction lies right in the middle between the baseline model and the model with 100% correct noun vectors.

The results of this model for the dataset after the *second annotation* demonstrate that the it has decent accuracy for each motion component. However, the fact that the model struggles with passive object motion prediction tells us that it is not quite clear for the model what it must learn.

Table 2.4: Motion prediction accuracy on the test set after the *automatic annotation*. The model was implemented with motion component classification, trained on 18,326 videos, and evaluated on 5,625 videos.

Motion components	***RGB***	***Flow***	***Fused***
Entire code	46.95	43.66	48.48
Interaction bits	87.96	84.68	86.83
Recursive active bit	88.07	85.53	89.05
Prismatic active bits	88.78	88.75	88.89
Revolute active bits	87.48	87.13	87.70
Recursive passive bit	97.65	97.19	97.58
Prismatic passive bits	67.79	64.94	68.60
Revolute passive bits	75.47	74.54	76.75
Hands bit	84.27	81.46	85.03
Tools bit	86.10	84.68	87.47

As we mentioned in Section 2.2.2, we used not the most optimal method for reducing the trajectory to a single bit, which is making it 0 if the objects' trajectory bits are identical and 1 otherwise. The best way to resolve this problem would be using the complete motion code format. In addition, the baseline model with no nouns performs poorly. If we wanted to rely purely on the visual data and predict the full-fledged motion code, we need a larger dataset, which is why we developed Word2Motion model and annotated the rest of the EPIC Kitchens dataset. After annotation, we split the dataset into train/validation/test sets using 60/20/20 split strategy. We also make the verb class distribution to be identical for each set. This is done because we wanted to use the same dataset and motion prediction model for the verb classification model.

Table 2.4 shows the performance of the motion prediction model on the test set after the *automatic annotation*. Unlike the previous model, this model predicts the complete version of the motion code. Also, this model does not use nouns at all. We can see that the model performs much better in comparison with the baseline from Table 2.3. This is especially impressive because the complete version of the motion code has about 6.5K valid outputs, which is significantly higher than 180 of the previous model. However, we should keep in mind that this model was trained

Table 2.5: Motion prediction accuracy on the entire dataset after the *second annotation*. The model was implemented with motion component classification, trained on 18,326 automatically annotated videos, and evaluated on 4,488 manually annotated videos.

Motion components	***RGB***	***Flow***	***Fused***
Entire code	21.99	19.47	22.28
Interaction bits	73.51	68.43	71.79
Recursive active bit	86.41	83.85	87.23
Prismatic active bits	69.45	67.94	68.58
Revolute active bits	69.65	67.07	68.85
Recursive passive bit	93.61	93.25	93.81
Prismatic passive bits	55.10	49.75	54.46
Revolute passive bits	62.86	58.07	62.05
Hands bit	76.47	73.64	76.89
Tools bit	83.18	80.57	83.93

with the motion code labels that were estimated with the Word2Motion model. That model was able to predict each component with at least 74% accuracy, but it was trained on the same dataset after the *second annotation*. In other words, in addition to imperfect estimation, the Word2Motion model estimated the motion codes for the dataset that is almost 10 times larger than its training set (3K vs. 28K). Therefore, the robustness of the results from Table 2.4 is arguable.

Nonetheless, we can still evaluate the model on the dataset after *second annotation*. The only problem is that some of those videos could have been used during the training of the model since we did not separate them from the entire EPIC Kitchens dataset, while the split process did not consider whether the video was in *second annotation* or not. Table 2.5 shows the results of using the model on the 4,488 videos that were manually annotated. We can see that the entire motion code prediction is much lower than from the Table 2.4. This demonstrates the impact of the noise in using the motion code labels estimated by Word2Motion model. Note that we used Word2Motion to annotate the entire dataset, including overwriting the manually annotated 4,488 videos. Despite such a low motion code estimation accuracy, the estimation of the individual motion components is still acceptable. The prismatic trajectory estimation of passive object is the lowest with 55%

accuracy. It is still better than always picking the value with the highest appearance rate (45%). These results demonstrate the importance of the robust data annotation. In the future work, we consider using both semantic and visual data to achieve better motion estimation for annotation. The results only prove that semantic data is not sufficient to represent motions. Additionally, we plan to ues more sophisticated word embedding methods. For instance, Embeddings from Language Models (ELMo) [3] encodes the words based on the sentence they were used in. They embed each word in the sentence via Word2Vec or GloVe and feed them to the bi-directional LSTM cells to obtain contextualized representation for each word.

2.5.3 Action Recognition

As with the motion prediction model, we conducted experiments after the *second annotation* and after the *automatic annotation*. For the former dataset, we train all models on 2,742 training videos, validate on 786 videos, and test on 1,517 videos. The test set was sampled from the remaining EPIC Kitchens videos that were not annotated with motion codes. For motion prediction M_x and motion prediction using nouns $M_{x,z}$ we use the models discussed in the previous section. We refer to the baseline verb classifier as V_x and to the baseline that also has direct access to nouns as $V_{x,z}$. We denote the final verb classifier that combines the outputs of the selected verb classifier (V_x or $V_{x,z}$) and motion prediction model (M_x, $M_{x,z}$) as $\hat{V}(V, M)$. It is important to note that if one of two models has direct access to nouns, then the other does not necessarily has the access to nouns as well. For instance, in the model $\hat{V}(V_x, M_{x,z})$, the baseline verb classifier V_x does not have access to nouns. Instead, only motion prediction model $M_{x,z}$ uses nouns during training and inference to achieve higher *motion prediction* performance. Hence, the knowledge of objects does not directly affect the performance of the entire $\hat{V}(V, M)$ model, but the application of nouns in the individual sub-models improves their individual performances.

Table 2.6: Verb classification accuracy on the validation set after the *second annotation*. The results are shown in top-1 accuracy. The models were trained on 2,742 videos and evaluated on 786 videos.

Methods	***RGB***	***Flow***	***Fused***
Baseline, V_x	41.60	39.82	45.04
Baseline with nouns, $V_{x,z}$	48.22	44.15	49.24
Predicted Motions, $\hat{V}(V_x, M_x)$	41.22	40.46	46.18
Predicted Motions with nouns, $\hat{V}(V_x, M_{x,z})$	43.13	42.11	47.20
Ground Truth Motions*, $\hat{V}(V_x, \bar{M}_x)$ (Using true motion code as embedding)	**53.82**	**53.69**	**57.63**

2.5.3.1 Results on Second Annotation Dataset

Table 2.6 shows the top-1 verb classification accuracy results of different models on the validation set. We use the validation set results to be able to compare all models with $\hat{V}(V_x, \bar{M}_x)$. This model combines the baseline verb classifier with the ground truth motion codes. Since, we did not have a separate test set annotated with motion codes, we used the validation set instead. We can see that the knowledge of nouns significantly improves the baseline, which is what we have already observed in the motion prediction model performance from Table 2.3. However, that performance gain cannot compete with integration of the ground truth motions. We see over 10% gain of $\hat{V}(V_x, \bar{M}_x)$ over the baseline, and more than 8% boost over the $V_{x,z}$. This observation shows that knowledge of objects is quite beneficial, but the huge gap between the semantic data and visual data does not let it go higher. Hence, more visually informative features, such as motion codes, provides more valuable information. However, we cannot assume that the model will always have access to ground truth nouns and motion codes. Moreover, we use these two models just to compare their impacts on verb classification. Therefore, we should also take a look at the models with the estimated motion codes. We can see that $\hat{V}(V_x, M_x)$ has very insignificant improvement over the baseline. However, the model $\hat{V}(V_x, M_{x,z})$ increases the accuracy even higher. We know that the difference between these two models comes from the application of nouns in the motion

Table 2.7: Verb classification accuracy on the test set. The results are shown in top-1 accuracy. The models were trained on 2,742 videos and evaluated on 1,517 test videos. We sampled test videos from the partition of EPIC Kitchens dataset that was not annotated during the *second annotation*.

Methods	***RGB***	***Flow***	***Fused***
Baseline, V_x	33.36	31.64	36.12
Predicted Motions, $\hat{V}(V_x, M_x)$	33.62	32.30	36.78
Predicted Motions with nouns, $\hat{V}(V_x, M_{x,z})$	**34.08**	**34.74**	**38.04**
Baseline with nouns, $V_{x,z}$	38.69	38.76	41.73
Predicted Motions, $\hat{V}(V_{x,z}, M_x)$	**38.89**	37.05	42.06
Predicted Motions with nouns, $\hat{V}(V_{x,z}, M_{x,z})$	38.83	**39.95**	**42.12**

estimation, and that nouns have positive impact on motion estimation as shown in Table 2.3. Considering these factors, we may conclude that better motion estimation is the key to the higher verb classification accuracy, which increases the value of the motion representation via motion codes.

Table 2.7 shows the results on the test set. One can observe that incorporating the predicted motions in the verb classification task consistently improves the overall accuracy. As in the validation set, we can see that augmenting the verb classifier with semantic information of the object-in-action significantly improves the performance of the baseline model. Despite such a noticeable jump, adding the estimated motion code with nouns ($M_{x,z}$) and without nouns (M_x) shows that motion features can still contribute to the model $V_{x,z}$, even though the motion estimation accuracy is not higher than 45%.

From both Table 2.6 and Table 2.7, we can observe that, in most cases, models that use optical flow frames benefit more from using motion codes than their RGB counterparts. In Table 2.7, the first baseline V_x classifies verbs more accurately when it uses RGB frames as opposed to using optical flow frames. After applying motions predicted by M_x, we can see that the optical flow model improved more than the RGB model, though still giving lower overall accuracy. However,

after applying an improved motion code estimator $M_{x,z}$, the performance of the flow model became better than the RGB model. In the baseline model with nouns $V_{x,z}$, RGB and flow models have very close performances, but model $\hat{V}(V_{x,z}, M_{x,z})$ performs noticeably better with flow frames. On validation set, both baseline verb classifiers, V_x and $V_{x,z}$, perform significantly better on RGB modality, but that distance gets reduced with motion code embedding and gets to its minimum when using ground truth motion labels as shown in Table 2.6. This observation demonstrated high correlation between our motion codes and optical flow vectors. Varol *et al.* have emphasized the importance of motion feature estimation for action recognition, including more accurate optical flow computation [22]. Therefore, we believe that our motion code embedding model has the potential to be a significant add-on feature extractor for action recognition that would provide explainable motion features.

2.5.3.2 Results on Automatically Annotated Dataset

The annotation via Word2Motion model let us use the whole EPIC Kitchens dataset. As the result, the test set has increased from 1,517 samples to 5,625 samples. In addition, the models classify all 125 verb classes instead of the previous 33 classes. However, we should keep in mind that the dataset is still very unbalanced with about 60% of all videos belonging to 4 unique classes. This time, we do not use any prior knowledge about the nouns, since the dataset is sufficiently large. As we implemented our baseline by following the NTU-CML-MiRA team's implementation for EPIC-KITCHENS Action Recognition Challenge 2019, we also include their reported results. We use the motion code estimator whose performance is reported in Table 2.4 on the same test set.

Table 2.8 shows that the application of the predicted motions is still beneficial for the top-1 verb classification accuracy. Nonetheless, we can see that margin of this improvement is not that significant, just like in the model trained on the previous dataset from Table 2.7. This is not

Table 2.8: Verb classification accuracy on the test set after the *automatic annotation*. The results are shown in top-1, top-3, and top-5 accuracy. The models were trained on 18,326 videos and evaluated on 5,625 test videos.

Methods	***RGB***	***Flow***	***Fused***
Top-1			
Baseline (reported), V_x^1	47.71	49.7	53.68
Baseline (ours), V_x	51.77	52.94	58.74
Predicted Motions, $\hat{V}(V_x, M_x)$	**52.43**	**53.49**	**58.93**
Top-3			
Baseline (ours), V_x	76.46	**76.96**	**82.47**
Predicted Motions, $\hat{V}(V_x, M_x)$	**77.60**	76.18	82.38
Top-5			
Baseline (ours), V_x	84.46	**84.30**	**89.16**
Predicted Motions, $\hat{V}(V_x, M_x)$	**84.96**	84.23	88.92

surprising, since the motion code annotations were quite noisy. Additionally, Table 2.4 shows that the motion code estimation is still lower than 50%, while Table 2.5 shows that using this model on manually annotated videos has about 20% accuracy. These numbers suggest that we need to have a more robust way to annotate videos with motion codes, which is in our plans for the future work. However, even with this motion code prediction performance, we are still getting some performance boost for verb classification. Table 2.8 also shows that our implementations have better accuracy than the reported accuracy for the same model with the same architecture, training process, and other hyper-parameters. This is probably due to the variations of our and their test sets. However, we made sure to precisely follow their dataset split strategy.

In addition to the more accurate video annotation, we are working on a more sophisticated method for combining the predicted motion codes into the verb classifier. One way can be the application of attention modules in the learning algorithm. For instance, we could use the predicted motion codes to generate attention maps that will attend important features from the final feature map. Another method is to use the output of the baseline as the input to the neural network that maps it to the motion codes. Hence, in addition to be able to classify verb classes, the final layer

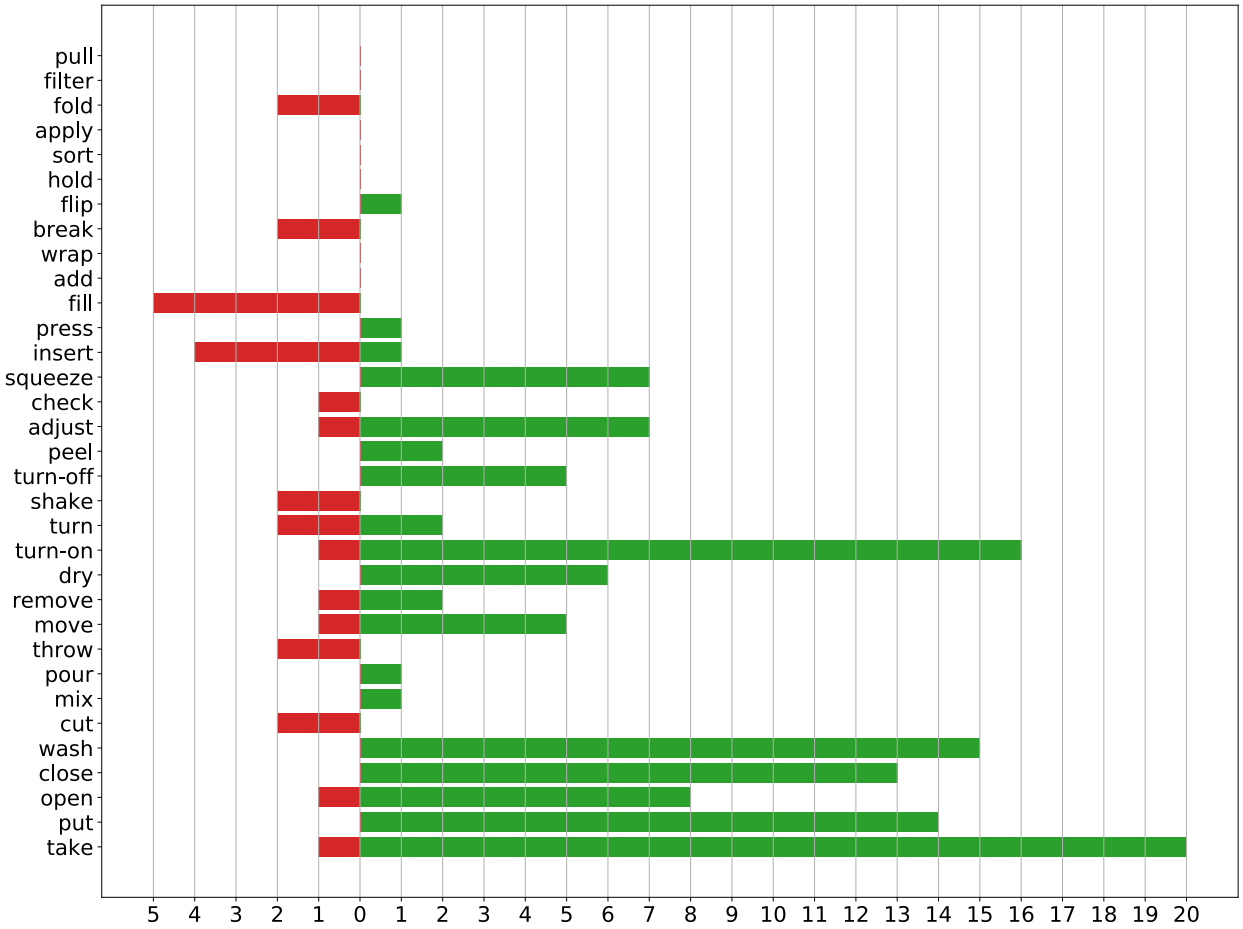


Figure 2.12: Number of videos that were correctly classified by $\hat{V}(V_x, \bar{M}_x)$, but not by V_x. Red bars show the opposite.

of the baseline must be able to predict the motion codes. Such kind of constraint may encourage the model to learn more salient visual features and we can balance the learning process by using different weights for two loss functions.

2.5.3.3 Qualitative Results

Figure 2.12 shows the number of videos that were incorrectly classified by the baseline model V_x and were corrected by our model that uses ground truth motion labels $\hat{V}(V_x, \bar{M}_x)$. It also shows videos that were misclassified by our model but were correct on the baseline. We can see that most of the classes benefit from using ground truth motion codes, with some exceptions. Five

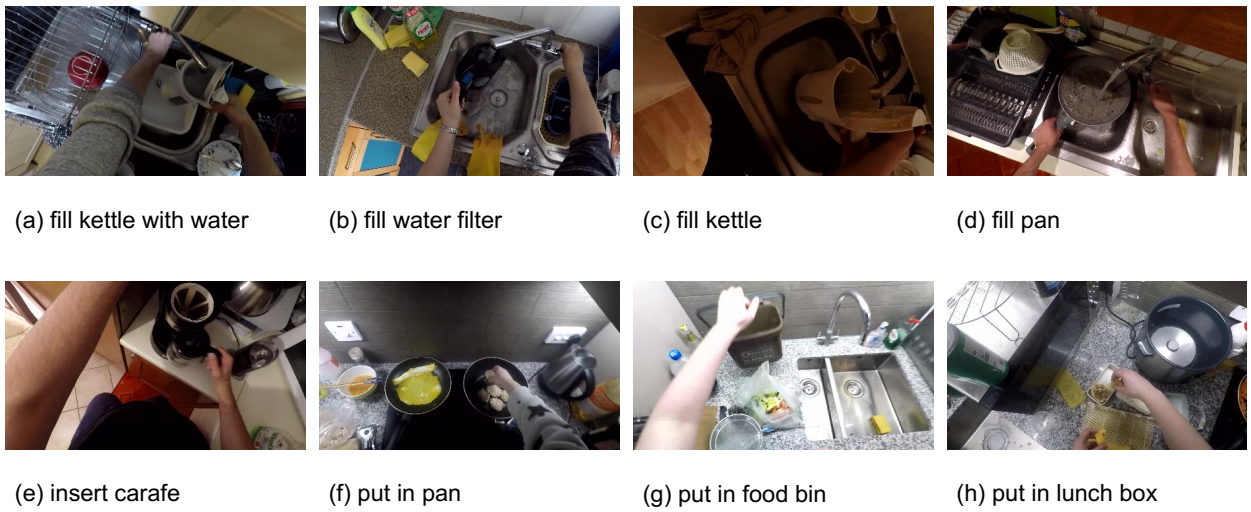


Figure 2.13: Videos from 2 classes that were incorrectly classified by our model. The top row shows the frames for class “fill" and the bottom row shows the frames from class “insert".

videos of class “fill" were predicted correctly with the baseline model and were misclassified by our model. Instead, they were classified as “put" (2 instances) or “take" (3 instances). By watching those videos, we noticed that motions executed on them were indeed closer to put or take, as all of them illustrate person holding a kettle, water filter, or pan under the tap while the tap is filling those containers with water. Figure 2.13 demonstrates sample frames of those videos with their annotated narrations.

The other class that suffered from our model is “insert" with 4 misclassified videos. What is interesting about those videos is that three out of four of them were narrated as “put-in" action, while the class key is called “insert". Our model classified these videos as “put" class and we, as humans, agreed on that after watching the videos as shown on the bottom row of Figure 2.13. All four videos show putting an object on the table, pan, or plate. Only video on Figure 2.13 (e) could be argued to be labeled as an “insert" rather than “put", where the actor places the glass carafe under the coffee maker. However, this is true from the semantic perspective and only from the assumption that the carafe is a part of the coffee maker setup. From the motion perspective,

it appears as if the actor is putting an object on a stand while the top part of the coffee maker is occluding the final segment of the motion.

In short, the results shown in Figure 2.12 and the analysis of videos that were misclassified by our model confirm that motion codes provide additional robustness to action recognition from the motion mechanics perspective. However, we assume that this is mostly due to the limitation of the way the verbs were grouped into classes. This problem could have been also avoided with other methods for fine-grained action recognition, such as our proposed method in Chapter 3. Had more fine-grained labels been used, the predicted labels would more adequately describe the action taking place.

Chapter 3: Action Recognition with Cross-Modal Embedding

In this chapter, we introduce our method for action recognition with application of embedding principles. Our framework was inspired by the work from [36], where Wray *et al.* use cross-modal embedding functions to map the visual data from the videos and the semantic data from the annotation sentences into a shared embedding space. The idea is to use the loss function that encourages the resulting embeddings to be close to each other if the video-text pairs match. Otherwise, it will push them further away. We found such a method to be suitable to be converted into the verb classification framework, that we can later compare to other action recognition frameworks. However, since the model is trained using the actual words narrated by the annotators, it learns more fine-grained representation, rather than the traditional classification models that use one-hot vector representations for the coarse-grained class groups.

3.1 Framework Description

Wray *et al.* implemented an effective cross-modal retrieval algorithm that they called Joint Part of Speech Embedding [36]. They first parse the video captions into separate parts of speech (e.g., verbs, nouns). After that, words are encoded via a word embedding model and video features are extracted with a deep convolutional neural network. Both features are further embedded into a shared space, such that relevant embedded vectors (i.e., they belong to the same class) are close to each other, while negative pairs are far away. The authors learned separate embedding functions for each part of speech and joined them together for fine-grained action retrieval.

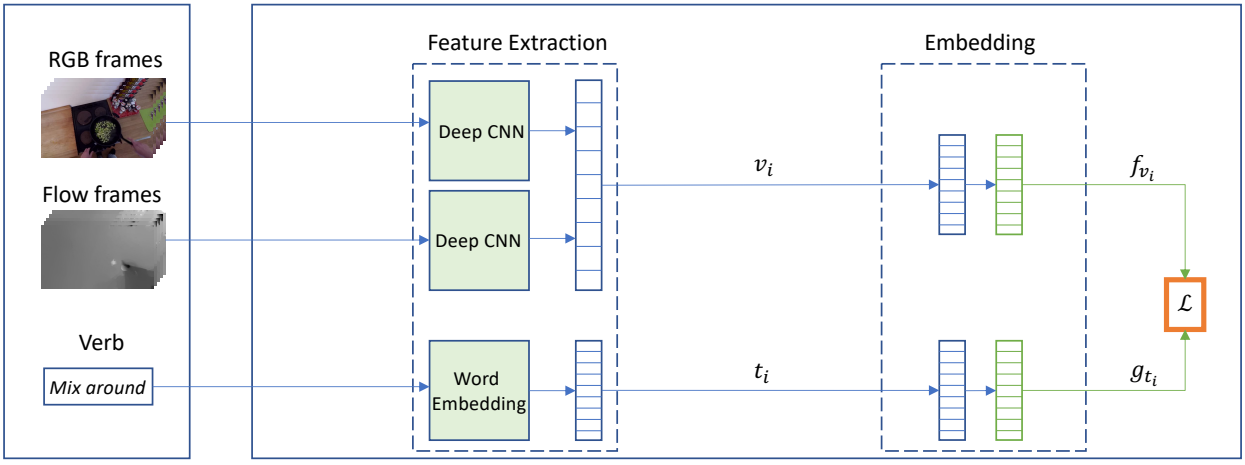


Figure 3.1: Our proposed framework. Two-Stream CNN model extracts the visual features from RGB and optical flow frames and concatenates them into a single visual vector. Word embedding model extracts the semantic features from words. The embedding functions map the features into a shared space.

We focused on a single part of speech, namely verbs. As in [36], we build a model that maps visual and text features to the shared space. Formally, let $\{(v_i, t_i)|v_i \in V, t_i \in T\}$ be a set of pairs, where v_i corresponds to the visual features from the video and t_i corresponds to the word-level embedding of the verb that the video demonstrates. We want to learn embedding functions $f : V \rightarrow \Gamma$ and $g : T \rightarrow \Gamma$, such that $f(v_i)$ and $g(t_i)$ will result in vectors that are close to each other. Both functions were implemented as 2 fully connected layers with ReLU activation. During evaluation, we predict the verb class by finding the verb whose embedding $g(t)$ has the smallest Euclidean distance to the embedding $f(v)$ of the current video clip. The structure of our framework is depicted in Figure 3.1.

To learn the embedding functions, we use a cross-modal pairwise ranking loss, also known as triplet loss. The idea is to encourage the model to make the embeddings in positive pairs more similar and embeddings in negative pairs less similar to each other. The loss function looks as follows:

$$L_{v,t} = \sum_{(i,j,k)\in\mathcal{T}_{v,t}} \max(d(f_{v_i}, g_{t_j}) - d(f_{v_i}, g_{t_k}) + \gamma, 0) \tag{3.1}$$

$$\mathcal{T}_{v,t} = \{(i,j,k)|v_i \in V, t_j \in T_{i+}, t_k \in T_{i-}\} \tag{3.2}$$

$$L_{t,v} = \sum_{(i,j,k)\in\mathcal{T}_{t,v}} \max(d(g_{t_i}, f_{v_j}) - d(g_{t_i}, f_{v_k}) + \gamma, 0) \tag{3.3}$$

$$\mathcal{T}_{t,v} = \{(i,j,k)|t_i \in T, v_j \in V_{i+}, v_k \in V_{i-}\} \tag{3.4}$$

The function $d(x_1, x_2)$ is the distance measurement between two vectors computed via Euclidean distance. In these equations, the value approaches zero if the distance between positive pairs is less than the distance between negative pairs by a constant margin γ. During training, $L_{v,t}$ is computed by summing all valid triplets in set $\mathcal{T}_{v,t}$, where T_{i+} represents set of relevant verbs and T_{i-} represents set of negative verbs to video v_i. Similarly for $L_{t,v}$.

The embedding space must be well-defined, and it will not be possible without having any constraints for embedding functions. Authors suggest adding within-modal triplet losses to preserve the structure of the neighborhood in the joint space. They use the following equations:

$$L_{v,v} = \sum_{(i,j,k)\in\mathcal{T}_{v,v}} \max(d(f_{v_i}, f_{v_j}) - d(f_{v_i}, f_{v_k}) + \gamma, 0) \tag{3.5}$$

$$\mathcal{T}_{v,v} = \{(i,j,k)|v_i \in V, v_j \in V_{i+}, v_k \in V_{i-}\} \tag{3.6}$$

$$L_{t,t} = \sum_{(i,j,k)\in\mathcal{T}_{t,t}} \max(d(g_{t_i}, g_{t_j}) - d(g_{t_i}, g_{t_k}) + \gamma, 0) \tag{3.7}$$

$$\mathcal{T}_{t,t} = \{(i,j,k)|t_i \in T, t_j \in T_{i+}, t_k \in T_{i-}\} \tag{3.8}$$

As previously, the vectors of videos that belong to the same semantic class must be closer, while pushing away negative pairs from each other. Similarly for the text embeddings. The final loss function is computed via a linear combination of all 4 loss functions:

$$L = \lambda_{v,t}L_{v,t} + \lambda_{t,v}L_{t,v} + \lambda_{v,v}L_{v,v} + \lambda_{t,t}L_{t,t} \tag{3.9}$$

In experiments, the weights of cross-modal loss functions are generally greater to focus more on cross-modal retrieval.

3.2 Experiments

3.2.1 Dataset

There are 125 verb classes in total in EPIC Kitchens dataset [48]. However, we only use a smaller subset of 26 classes that contain at least 100 clips in the dataset, leaving us with 26,710 clips in total. This modification was done because classes with less than 100 clips would have a negligible effect on the training and evaluation of our model. In addition to that, their elimination reduces the size of the entire dataset by only 6%. The resulting dataset was split into train, validation, and test sets following a 60/20/20 split strategy. The class distributions among all 3 sets are identical

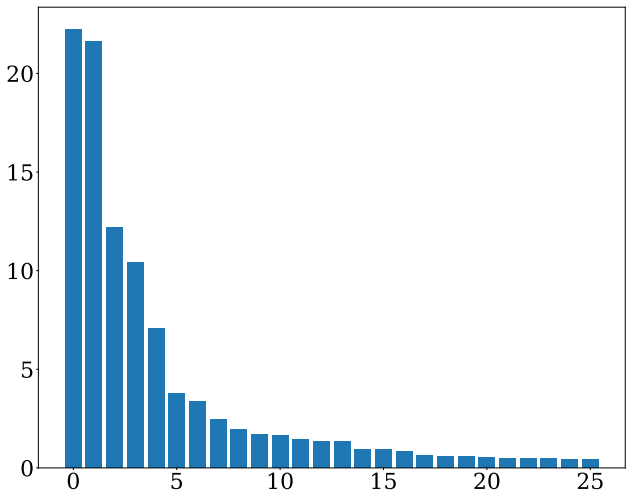


Figure 3.2: Verb class distribution in the dataset for many-shot verbs. Horizontal axis represents the verb class labels. Vertical axis represents the percentage of the class in the dataset.

and look as in Figure 3.2. During training, we use the actual verbs that were narrated for each video, while for testing we use the verbs that represent the class keys.

It is important to mention that the classes may refer to multiple verbs as some words have identical semantics. Table 3.1 shows that one class may encompass more than 10 different verbs. In fact, the class "put" has 30 different verbs. One could suggest that such kind of variation in words can be a big challenge for the classification with cross-modal embedding. For instance, the video that was labeled with verb "grab" must be classified with class "take", but because during training we embedded the word "grab", it will be less likely classified as "take". On the other hand, traditional classifiers with one-hot vector representations are secured from such vulnerability. In reality, this variation of words should not cause significant damages for classification with embedding because the words are preliminarily encoded with a word embedding model. Modern word embedding models, such as Word2Vec [1], were pre-trained on a large text corpus to be able to encapsulate the context for each word. Therefore, words that may look different but have similar semantics will be relatively close to each other, eliminating potentially negative impact on the classification problem. In addition to that, we consider a pair verbs from the same verb class as

Table 3.1: Set of verbs that were associated with each verb class. Some verbs from one class are similar to the verbs in other classes (e.g., *put-in* from class *insert* is close to verbs in class *put*).

Class Name	***Verbs in the Class***
put	put, pose, put-away, place-that, place-down, put, set-down, put-over, place-on, return, put-on, place-away
take	take, grab, pick, draw, get, grab-up, collect-from, take-up, grab-down, pull-down, fetch, pick-up
wash	wash, sponge, lather, wash-with, rinse, rinse-off, soap-up, wash-off, soap, rise, wash-up, clean-around, clean-off
open	open, unzip, open-up
close	close, close-off, shut
cut	cut, chop, chop-off, cut-off, slice-into, slice-along, slice-off, stem, slice-up, cut-into, dice, half, halve, chop-up
mix	mix, beat, mix-around, stir-with, whisk, stir, blend, mix-in, stir-in
pour	pour, pour-in, tip-in, pour-out, pour-on, pour-into, sieve
move	move, transfer, move-around
turn-on	turn-on, start, begin, ignite, switch-on, activate, water-on, play, start-to, restart, light
remove	remove, extract, take-off, remove-out, take-out, get-out, remove-inside, remove-from, pick-out
turn-off	turn-off, switch-of, water-off, switch-off, switch-out, shut-off, turn-of
throw	throw, throw-out, recycle, dispose-of, throw-over, throw-away, throw-in, bin, trow, toss
dry	dry, dry-off, towel
peel	peel, skin-from, skin, peel-off, peel-back
insert	insert, put-in, put-into, fit, place-in, put-inside
turn	turn, rotate
shake	shake, shake-off, shake-out
squeeze	squeeze, squidge, squidge-into, squash, squish-into, wring-out, wring, squish, squeeze-into
press	press, push-down, collapse, compress, push, press-on
check	check, ensure, test, look-in, watch, inspect, check-on
scoop	scoop, spoon-in, spoon, scoop-out, scoop-up
empty	empty
adjust	adjust, change, regulate
fill	fill, fill-with, fill-up, stuff
flip	flip, overturn, turn-over

a positive pair during the computation of a triplet loss. This means that the embedding function $g : T \rightarrow \Gamma$ will learn to encode the words "grab" and "take" to be very close to each other.

3.2.2 Training Details

One of the biggest challenges in training the model from Figure 3.1 is that the fusion of RGB and optical flow features happens in the middle of the pipeline. In contrast, most of the existing two-stream methods employ late-fusion strategy, thus allowing them to train RGB and optical flow models separately. To be able to leverage the performance boost from optical flow features, we first decided to separately extract visual feature vectors from both RGB and flow frames. We used the TSN-ResNet50 model that was pre-trained on the EPIC Kitchens dataset [57] (i.e., the target dataset). The model uniformly splits the video into 8 segments and randomly selects one RGB frame and 5 consecutive flow frames from each segment. Both output vectors are 2048-dimensional.

The biggest drawback of the method mentioned above is the fact that we eliminate the possibility to tune the convolutional layers for our model. Visual content has great discriminative capabilities due to its high dimensionality and we have deprived ourselves of the opportunity to take advantage of it. On the other side, training two deep CNNs in parallel is computationally too expensive. Therefore, in addition to the previously described method, we decided to implement end-to-end training by eliminating the optical flow model and only using the RGB model. From each video, we sampled 12 frames by segmenting video into 12 segments and randomly selecting a frame from each segment.

We tried to use the hyper-parameters that were recommended by Wray *et al.*, but not all of them were fulfilled. For instance, authors randomly select 100 triplets to compute the loss value for one batch. We were not able to do that with TensorFlow [53] because any random operation results in the loss of the gradients. Instead, we take the average value over the entire set of valid triplets within the batch. Besides that, the authors use batches with 256 samples. When training with pre-computed visual features, we were able to use a batch size of 256 samples, but for a model

Table 3.2: Micro and macro F1 scores for experiments with pre-computed visuals. CM:WM column represents the ratio of weights for cross-modal and within-modal loss functions.

Oversampling	CM:WM	FC Units	Macro F1 (%)	Micro F1 (%)
False	7:3	[1000, 500, 32]	5.11	17.92
False	7:3	[1000, 500, 32]	5.04	14.16
True	1:1	[3000, 500, 32]	5.03	15.35
True	7:3	[1000, 500, 32]	4.92	15.52
True	7:3	[1000, 500, 32]	4.77	15.43
False	7:3	[1000, 500, 32]	4.71	16.49
False	7:3	[3000, 500, 32]	4.54	17.06
False	1:1	[3000, 500, 32]	3.94	15.06

with I3D, we could not get higher than 64. In the paper, the authors did not specify the value for the constant margin γ in the loss function. After some observations, we found that the most optimal value was equal to 0.5. Finally, Wray *et al.* trained the model step-wise, while we decided to perform the training based on epochs. That way we were able to run the evaluation of the model on the validation set by performing verb classification and saving the weights with the highest accuracy. The remaining hyper-parameters were set as recommended in [36]. Specifically, we set the dimensionality of the embedding space to 256 and trained the model with Adam optimizer. The weights for the cross-modal loss functions were set to be ten times higher than the within-modal functions, 1.0 and 0.1 respectively. As in [36], we used a 100-dimensional Word2Vec model [1] that was pre-trained on Wikipedia [54, 55, 56]. Authors state that other word embedding models provide identical results.

Both methods were trained for 50 epochs. The training session of the model with pre-computed visual features took less than 10 minutes and the model with I3D took more than 2.5 days on a single Titan V GPU.

3.3 Results and Discussion

In this section, we declare the results of our model using 2 previously mentioned methods.

3.3.1 Pre-Computed Visual Features

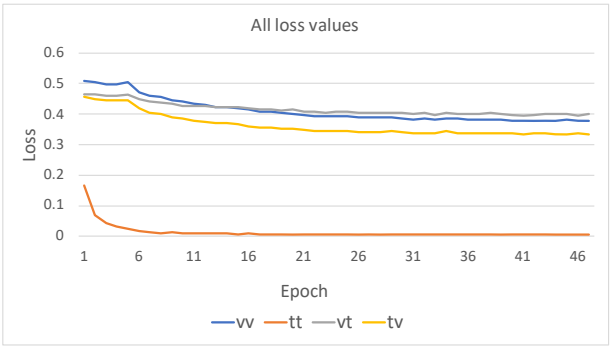
As was mentioned in Section 3.2.2, by pre-computing visual features we lost the ability to tune the deep CNN model. In addition to that, the dataset itself is quite unbalanced, as shown in Figure 3.2. Due to these reasons, we were not able to avoid serious overfitting to the training set, which resulted in the model classifying every video with the first class (i,e., 23% accuracy). The value of the loss function was not stable and it was constantly jumping up and down. The application of regularization techniques, such as L2 regularization [52] and dropout [58], did not benefit the model at all. We also tried to create separate TensorFlow Dataset objects for each class, which allowed us to feed the model with batches with a balanced distribution of classes. This data balancing method resulted in better F1 score (about 5%), but in lower accuracy (at most 19%). When we tried to train the model with conventional one-hot vectors, the results were identical.

Table 3.2 shows the obtained macro and micro F1 scores from additional experiments with and without data balancing and with different ratios of weights for cross-modal and within-modal loss functions. For oversampling, we used SMOTE algorithm introduced by Chawla *et al.* [59]. The column *FC Units* shows the number of fully connected layers and units. For example, the best model had 3 fully connected layers, with 1000, 500, and 32 hidden units respectively.

3.3.2 I3D Model

The disappointing results of the previous method made us switch the direction towards the training the model without optical flow features, but with the ability to tune the convolutional filters. We were still not able to train all layers of I3D, so we relaxed only the last two 3D inception units. Nevertheless, the results were much better than in the previous method.

As we can see from Figure 3.3a, the values of the loss functions decrease persistently. There is not a lot of dynamics in the figure as the metrics are shown based on the average value in one



(a) The values of loss functions for mapping video-to-video (vv), text-to-text (tt), video-to-text (vt), and text-to-video (tv) embeddings after each epoch on training set.

(b) The value of the total loss at each epoch on training set. The blue line represents the actual value. The orange line represents the value without the constant margin γ.

Figure 3.3: Loss function values on the validation set.

epoch, which encapsulates more than 250 batch steps. The value of the text-to-text loss was the lowest, which is not surprising given the fact that text data is much less dimensional than a video clip. We can also see that both text-to-video and video-to-video losses follow the identical trend and they are both better than video-to-text loss. Interestingly, video-to-video loss initially started in the worse position than video-to-text, but it was able to improve and outperform the latter. This can be explained by the fact that the variation of video features is much higher, and they are much more discriminative. Thus, the differences between positive and negative video samples for each anchor could be significant enough to let the embedding function to map them further away. This could be more complicated for the video-to-text case, where the text features may not vary enough due to the small set of unique verbs in the dataset. Despite that, the value of the video-to-text loss, as well as all other losses, is lower than the constant margin γ that was set to 0.5. This means that on average, the difference between irrelevant features is higher than the difference between relevant ones.

The total loss function also constantly decreases, as shown in Figure 3.3b. We can see that it is always less than zero without the margin. It could be argued that this is mostly because of

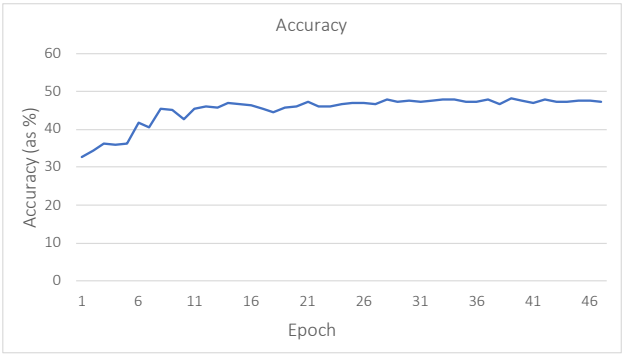


Figure 3.4: The top-1 accuracy of the model on the validation set. The results are computed right after each training epoch.

the exceptional performance of text-to-text embedding. However, this is not the case as the weight of text-to-text loss is ten times lower than cross-modal losses. In addition to that, the curve of text-to-text loss in the range of epochs 1 and 6 is quite steep, while the curve of the total loss is more similar to losses with the video target (i.e., $L_{v,v}$ and $L_{t,v}$).

Figure 3.4 shows that the dynamic of how accuracy changed is similar to the dynamics of the loss functions. The figure also demonstrates that the ability to tune the convolutional layers provides a significant performance boost within a single epoch of a training session. With over 30% accuracy, one training epoch already outperforms the previous model that uses pre-computed visual features (23% accuracy) by almost 10%. The accuracy stops to increase somewhere between epochs 35 and 40. We use the weights with the highest validation accuracy for the testing.

3.3.3 Comparison with Action Recognition Models

We compare our model with the results that were provided by the authors of EPIC Kitchens in [57]. They compare different state-of-the-art action recognition models, namely TSN, TSM, TRN, and M-TRN, that use 2 types of backbone CNN models, batch-normalized Inception and ResNet-50. Unfortunately, they did not benchmark the dataset on I3D. Nevertheless, we may refer to the results from [60], where the team from Baidu-UTS reports their results on EPIC Action Recognition

Table 3.3: Verb classification accuracy results on the testing videos. We provide top-1 and top-5 results for all models except the I3D NTU-CML-MiRA. All results are from RGB models.

Model	***Top-1***	***Top-5***
TSN BN-Inception	47.9	87.0
TSN ResNet-50	49.7	**87.2**
TRN BN-Inception	58.2	87.1
TRN ResNet-50	58.8	86.6
M-TRN BN-Inception	57.6	86.9
M-TRN ResNet-50	**60.1**	87.1
TSM ResNet-50	57.8	87.1
I3D BN-Inception (Baidu-UTS)	59.3	82.7
I3D BN-Inception (NTU-CML-MiRA)	47.7	—
Ours	48.2	78.0

Challenge 2019 using I3D and achieving the highest accuracy in the competition. We additionally provide results of other team from the same competition, namely NTU-CML-MiRA. The model of the team Baidu-UTS is more powerful than ours, since they sample 64 frames from a single video. Meanwhile, NTU-CML-MiRA team's model, just like ours, samples 12 frames per video. Therefore, we consider their results more relevant to our framework. We only consider the models for RGB frames and use top-1 and top-5 accuracy for metrics. The results are shown in Table 3.3.

Table 3.3 shows that our model outperforms the TSN with BN-Inception and I3D NTU-CML-MiRA by a small margin. The remaining models significantly outperform our model in top-1 accuracy and all models do much better than ours in top-5 accuracy. Considering the unconventional method for classification that our model implements, the obtained results show that the application of cross-modal embedding concepts in action recognition can impose a decent competition on current state-of-the-art models. In addition, if we consider the I3D NTU-CML-MiRA model as the closest counterpart in terms of implementation, we can see the our method actually outperforms it. We assume that with more action classes, our model could have shown much better results relative to conventional methods. Our model learns in a more fine-grained fashion and is more capable to predict the actual words that were mapped to the video. Meanwhile,

models with one-hot vector representations are limited to the coarse-grained set of classes that were defined by the annotator, which also makes the class groups subjective.

3.4 Conclusion

To conclude, we trained the model using a cross-modal embedding strategy and evaluated its performance in the action classification task. We were able to show that this method demonstrates results comparable to the current state-of-the-art action classification models, even though it was not specifically trained for this task.

For future work, we could incorporate embedding nouns for embedding object semantics. This modification would allow us to compare this model with the full-fledged action recognition models. Another direction could be unrolling the verb classes in the EPIC Kitchens dataset into more fine-grained classes and train conventional action classification models with the new labels. In this case, we could expect a better performance of our model compared to these models as they would most likely suffer from more classes.

Chapter 4: Conclusion and Future Work

In this thesis, we introduce the *motion taxonomy*, a hierarchical structure that defines motion attributes, such as interaction type, object trajectories, and their structural outcomes. We annotated the EPIC Kitchens dataset and train a motion prediction model that may act as a high-level motion embedding model for action recognition frameworks alongside with the optical flow vectors. Throughout our experiments, we demonstrate that accurately predicted motion features can significantly boost the performance of the baseline model. We also show that the motion codes provide more visually salient features than the knowledge about the objects-in-action. In addition to the motion prediction model, we developed a Word2Motion model, which allowed us to annotate the entire EPIC Kitchens dataset automatically. In the future work, we will try to improve the accuracy of the automatic annotations by including the visual data and more sophisticated sentence embedding methods. We believe that more accurate motion annotations on a large-scale dataset will provide us with even better motion prediction results.

We also introduce an action recognition model based on cross-modal embedding concepts. We demonstrate that this unconventional approach can compete with the state-of-the-art action recognition. At the same time, our method eliminates the drawbacks of the tradition frameworks that utilize one-hot vector representation for their labels. In the future work, we can decompose the verb classes in the EPIC Kitchens dataset into more fine-grained actions, and test whether our method will be more resistant to the increased difficulty level than the baseline counterparts. We can also learn the embedding space for nouns to achieve the full-fledged action recognition

framework. Finally, we will incorporate motion codes from motion prediction model and test their impact on performance.

Both frameworks provide us with valuable pieces of action representation. The motion prediction model extracts high-level motion features that eliminates the mechanical ambiguity. The action recognition via cross-modal embedding model extracts visual features that can be used to find fine-grained semantic labels represented via word-level embedding, which eliminates the semantic ambiguity of one-hot vectors and action classes. The combination of their outputs is a more informative and detailed action representation that can be used to precisely visual the actions without watching the video.

References

[1] Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. In *Advances in neural information processing systems*, pages 3111–3119, 2013.

[2] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*, pages 1532–1543, 2014.

[3] Matthew E Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. Deep contextualized word representations. *arXiv preprint arXiv:1802.05365*, 2018.

[4] Yongqiang Huang, Matteo Bianchi, Minas Liarokapis, and Yu Sun. Recent data sets on object manipulation: A survey. *Big data*, 4(4):197–216, 2016.

[5] Yongqiang Huang and Yu Sun. A dataset of daily interactive manipulation. *The International Journal of Robotics Research*, 38(8):879–886, 2019.

[6] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Label-embedding for attribute-based classification. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 819–826, 2013.

[7] Zeynep Akata, Florent Perronnin, Zaid Harchaoui, and Cordelia Schmid. Label-embedding for image classification. *IEEE transactions on pattern analysis and machine intelligence*, 38(7):1425–1438, 2015.

[8] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Learning to detect unseen object classes by between-class attribute transfer. In *2009 IEEE Conference on Computer Vision and Pattern Recognition*, pages 951–958. IEEE, 2009.

[9] Christoph H Lampert, Hannes Nickisch, and Stefan Harmeling. Attribute-based classification for zero-shot visual object categorization. *IEEE transactions on pattern analysis and machine intelligence*, 36(3):453–465, 2013.

[10] Jingen Liu, Benjamin Kuipers, and Silvio Savarese. Recognizing human actions by attributes. In *CVPR 2011*, pages 3337–3344. IEEE, 2011.

[11] David Paulius and Yu Sun. A survey of knowledge representation in service robotics. *Robotics and Autonomous Systems*, 118:13–30, 2019.

[12] David Paulius, Yongqiang Huang, Roger Milton, William D Buchanan, Jeanine Sam, and Yu Sun. Functional object-oriented network for manipulation learning. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2655–2662. IEEE, 2016.

[13] David Paulius, Ahmad B Jelodar, and Yu Sun. Functional object-oriented network: Construction & expansion. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1–7. IEEE, 2018.

[14] Shaogang Ren and Yu Sun. Human-object-object-interaction affordance. In *2013 IEEE Workshop on Robot Vision (WORV)*, pages 1–6. IEEE, 2013.

[15] Yu Sun, Shaogang Ren, and Yun Lin. Object–object interaction affordance learning. *Robotics and Autonomous Systems*, 62(4):487–496, 2014.

[16] Yu Sun and Yun Lin. Modeling paired objects and their interaction. In *New Development in Robot Vision*, pages 73–87. Springer, 2015.

[17] Ahmad Babaeian Jelodar, David Paulius, and Yu Sun. Long activity video understanding using functional object-oriented network. *IEEE Transactions on Multimedia*, 21(7):1813–1824, 2018.

[18] Karen Simonyan and Andrew Zisserman. Two-stream convolutional networks for action recognition in videos. In *Advances in neural information processing systems*, pages 568–576, 2014.

[19] Limin Wang, Yuanjun Xiong, Zhe Wang, and Yu Qiao. Towards good practices for very deep two-stream convnets. *arXiv preprint arXiv:1507.02159*, 2015.

[20] Christoph Feichtenhofer, Axel Pinz, and Andrew Zisserman. Convolutional two-stream network fusion for video action recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1933–1941, 2016.

[21] Limin Wang, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang, and Luc Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, pages 20–36. Springer, 2016.

[22] Gül Varol, Ivan Laptev, and Cordelia Schmid. Long-term temporal convolutions for action recognition. *IEEE transactions on pattern analysis and machine intelligence*, 40(6):1510–1517, 2017.

[23] Du Tran, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani, and Manohar Paluri. Learning spatiotemporal features with 3d convolutional networks. In *Proceedings of the IEEE international conference on computer vision*, pages 4489–4497, 2015.

[24] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[25] David Paulius, Yongqiang Huang, Jason Meloncon, and Yu Sun. Manipulation motion taxonomy and coding for robots. In *International Conference on Intelligent Robots and Systems (IROS)*, pages 1–6. IEEE, 2019.

[26] David Paulius, Nicholas Eales, and Yu Sun. A motion taxonomy for manipulation embedding. In *Proceedings of Robotics: Science and Systems*, Oregon, USA, July 2020. Accepted.

[27] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[28] Jônatas Wehrmann, Douglas M Souza, Mauricio A Lopes, and Rodrigo C Barros. Language-agnostic visual-semantic embeddings. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 5804–5813, 2019.

[29] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[30] Kyunghyun Cho, Bart Van Merriënboer, Caglar Gulcehre, Dzmitry Bahdanau, Fethi Bougares, Holger Schwenk, and Yoshua Bengio. Learning phrase representations using rnn encoder-decoder for statistical machine translation. *arXiv preprint arXiv:1406.1078*, 2014.

[31] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

[32] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

[33] Hao Wang, Doyen Sahoo, Chenghao Liu, Ee-peng Lim, and Steven CH Hoi. Learning cross-modal embeddings with adversarial networks for cooking recipes and food images. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 11572–11581, 2019.

[34] Chi Zhan, Dongyu She, Sicheng Zhao, Ming-Ming Cheng, and Jufeng Yang. Zero-shot emotion recognition via affective structural embedding. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1151–1160, 2019.

[35] Iro Laina, Christian Rupprecht, and Nassir Navab. Towards unsupervised image captioning with shared multimodal embeddings. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 7414–7424, 2019.

[36] Michael Wray, Diane Larlus, Gabriela Csurka, and Dima Damen. Fine-grained action retrieval through multiple parts-of-speech embeddings. In *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.

[37] Antoine Miech, Dimitri Zhukov, Jean-Baptiste Alayrac, Makarand Tapaswi, Ivan Laptev, and Josef Sivic. Howto100m: Learning a text-video embedding by watching hundred million narrated video clips. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 2630–2640, 2019.

[38] Meera Hahn, Andrew Silva, and James M Rehg. Action2vec: A crossmodal embedding approach to action learning. *arXiv preprint arXiv:1901.00484*, 2019.

[39] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR09*, 2009.

[40] Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

[41] Andrej Karpathy, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar, and Li Fei-Fei. Large-scale video classification with convolutional neural networks. In *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[42] Jeffrey Donahue, Lisa Anne Hendricks, Sergio Guadarrama, Marcus Rohrbach, Subhashini Venugopalan, Kate Saenko, and Trevor Darrell. Long-term recurrent convolutional networks for visual recognition and description. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2625–2634, 2015.

[43] Hakan Bilen, Basura Fernando, Efstratios Gavves, Andrea Vedaldi, and Stephen Gould. Dynamic image networks for action recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3034–3042, 2016.

[44] Amin Ullah, Jamil Ahmad, Khan Muhammad, Muhammad Sajjad, and Sung Wook Baik. Action recognition in video sequences using deep bi-directional lstm with cnn features. *IEEE Access*, 6:1155–1166, 2017.

[45] Jue Wang, Anoop Cherian, Fatih Porikli, and Stephen Gould. Video representation learning using discriminative pooling. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1149–1158, 2018.

[46] Khurram Soomro, Amir Roshan Zamir, and Mubarak Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[47] Hildegard Kuehne, Hueihan Jhuang, Estíbaliz Garrote, Tomaso Poggio, and Thomas Serre. Hmdb: a large video database for human motion recognition. In *2011 International Conference on Computer Vision*, pages 2556–2563. IEEE, 2011.

[48] Dima Damen, Hazel Doughty, Giovanni Maria Farinella, Sanja Fidler, Antonino Furnari, Evangelos Kazakos, Davide Moltisanti, Jonathan Munro, Toby Perrett, Will Price, and Michael Wray. Scaling egocentric vision: The epic-kitchens dataset. In *The European Conference on Computer Vision (ECCV)*, September 2018.

[49] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[50] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[51] Christopher Zach, Thomas Pock, and Horst Bischof. A duality based approach for realtime tv-l 1 optical flow. In *Joint pattern recognition symposium*, pages 214–223. Springer, 2007.

[52] Anders Krogh and John A Hertz. A simple weight decay can improve generalization. In *Advances in neural information processing systems*, pages 950–957, 1992.

[53] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dandelion Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. Software available from tensorflow.org.

[54] Ikuya Yamada, Akari Asai, Jin Sakuma, Hiroyuki Shindo, Hideaki Takeda, Yoshiyasu Takefuji, and Yuji Matsumoto. Wikipedia2vec: An efficient toolkit for learning and visualizing the embeddings of words and entities from wikipedia. *arXiv preprint 1812.06280v3*, 2020.

[55] Ikuya Yamada, Hiroyuki Shindo, Hideaki Takeda, and Yoshiyasu Takefuji. Joint learning of the embedding of words and entities for named entity disambiguation. In *Proceedings of The 20th SIGNLL Conference on Computational Natural Language Learning*, pages 250–259. Association for Computational Linguistics, 2016.

[56] Ikuya Yamada and Hiroyuki Shindo. Neural attentive bag-of-entities model for text classification. *arXiv preprint arXiv:1909.01259*, 2019.

[57] Will Price and Dima Damen. An evaluation of action recognition models on epic-kitchens. *arXiv preprint arXiv:1908.00867*, 2019.

[58] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[59] L.O. Hall N.V. Chawla, K.W. Bowyer. Smote: Synthetic minority oversampling technique. *J. Artificial Intelligence Research, vol. 16 pp. 321-357*, 2002.

[60] Xiaohan Wang, Yu Wu, Linchao Zhu, and Yi Yang. Baidu-uts submission to the epic-kitchens action recognition challenge 2019. *arXiv preprint arXiv:1906.09383*, 2019.