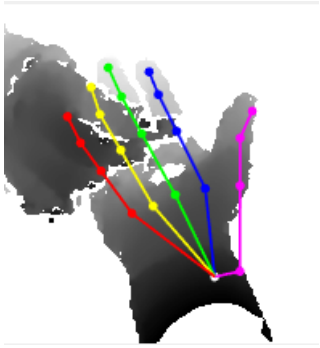
Imperial College London

MEng Final Year Project Report 2018

Imperial College Computer Vision and Learning Laboratory

Department of Electrical and Electronic Engineering

Action group recognition from 3D hand poses



Author:
Nathan Cavaglione

CID:
00980031

Course:
EE4T

Project supervisor:
Dr. Tae-Kyun Kim

Second Marker:
Krystian Mikolajczyk

June 20, 2018

Abstract

In this work, the effectiveness of grouping classes of actions to improve group and individual recognition accuracy is evaluated. Using the Imperial College Computer Vision Laboratory (ICVL) hand-object pose sequence dataset, 45 separate actions are manually grouped into super-classes each containing a few actions. At first, a Recurrent Neural Network (RNN) 'groupnet' is trained to identify the super-classes while RNN 'subnets' are trained to differentiate the individual actions inside each super-class. Subnets achieve high recognition accuracy (85%) while the groupnet remains the limitation (70%). Hence, a Variational autoencoder (VAE) is then used to embed the temporal sequences into a single 'thought vector' on which Kmeans clustering is applied. This attempts to automatically discover more discriminative cluster of super-classes built on insightful latent information.

Acknowledgements

This thesis has been supervised by professor Tae-Kyun Kim and post-doctoral researcher Guillermo Garcia-Hernando. They are both member of the Imperial College Computer Vision and Learning Lab (ICVL). Guillermo led my research on a day-to-day basis by allowing time to reflect with me on experiments and guiding me through any research direction decisions to take. Professor Kim met us regularly to supervise my progress throughout this final year project. I would like to express my sincere gratitude to both of them.

Contents

Chapter 1

Introduction

A human action can be seen as an ensemble of spatio-temporal trajectories that describe human motion [7]. Recently introduced cost-effective depth sensors and reliable real-time body pose estimation have added two new channels to the usual color (RGB) stream for action recognition. This has allowed skeleton-based action recognition to exist and be appreciated for its accuracy and efficiency [18]. In addition, with the recent introduction of wearable cameras, a new chapter in computer vision called egocentric vision has emerged where the user is the center of the action. A distinctive characteristic of this new paradigm relative to the classic third-person vision is that hands are very present in the scene [37]. On an economic perspective, the Virtual and Advanced Reality industry, where action recognition is largely used, is predicted to multiply by 4 its revenues between 2018 and 2020 [10]. All this makes action recognition an emerging field with some of the latest breakthrough research dating from 2017.

1.1 Objectives

The aim of this project is threefold. The goals are to (1) improve instance action recognition accuracy (2) by improving group recognition accuracy, (3) thereby gaining insight on the structure of hand action pose visualization sequences.

1.2 Ethical, Legal and Safety Plan

The Ethical, Legal and Safety Plan must detail what are the issues that are relevant to the project. This requires to check the project for issues in these areas, even though most projects have no such issues. First, safety-wise, the project is purely software based and, as a result, there can be no issues that can endanger one's physical integrity. Then, concerning ethics, there is no personal data or user testing involved in this research project up to now. Lastly, for legal issues, the deliverable is not expected to be commercialized, is not hardware related, and to the best of my knowledge, does not infringe any patents.

Chapter 2

Background

This part includes material that will help towards understanding the rest of the report. Experts may skip its reading.

2.1 Concepts

Most concepts are bookwork and are very well explained in the links provided.

2.1.1 Recurrent Neural Networks

Recurrent Neural networks are neural networks adapted to time series inputs. This blog post explains extensively their architecture involving LSTM cells [1] and this website [20] details their usefulness.

2.1.2 Variational Autoencoder

The original VAE paper is in the bibliography [21] but it is advised to read this blog post [15] to get an intuition of it. This paper [9] is also useful for an in-depth mathematical analysis and understanding of what is the 'reparametrization trick' that allows the VAE to use back-propagation.

2.1.3 Kmeans

The classic steps of k-means are:
1. Randomly initialise cluster centers (called 'centroids').
2. Assign data points to the closest centroid using Nearest Neighbour matching and the Squared Euclidean distance metric.
3. Recompute the centroid of each cluster as the mean of the points belonging to that cluster.
4. Repeat assigning data points and computing centroids until no data point changes clusters.
5. Return the coordinates of the set of centroids.

Other distance metrics are not considered as we are not guaranteed to find optimum centroids or partitions with them . Indeed, k-means is implicitly based on pairwise Euclidean distances between data points, because the sum of squared deviations from centroid is equal to the sum of pairwise squared Euclidean distances divided by the number of points. It is a multivariate mean in euclidean space. and non-euclidean distances will generally not span euclidean space. This is why k-means is for euclidean distances only.

Overall, k-means is simple and guaranteed to converge. However, it imposes on the user to pick K, is sensitive to initialisation and outliers, and is only limited to finding spherical clusters. It also scales badly in computation time when quantity of data increases ($O(n\hat{2})$) as it uses nearest neighbour matching. In high dimensions, the latter has difficulties to converge as concentration of measure tends to ensure that there are many centers with similar distances to any given point.

2.2 Dataset description

The dataset is fully described in the research paper introducing it [17]. Briefly, the hand pose data is taken with 21 magnetic sensors measuring movement in 3 directions (x,y,z) on the subject's hand. No sequence is longer than 170 frames and every sequence is taken at 30 frames per seconds. There is a total of 45 different action classes. Lastly, the data is preprocessed to compensate for anthropomorphic and viewpoint differences by normalizing poses to have the same distance between pairs of joints and defining the wrist as the center of coordinates

2.3 Interim report literature review

2.3.1 On the action recognition research at Imperial College

The project's title was initially named "Learning discriminative temporal transitions for action recognition and detection". It was the title of a recent research paper [18] published by the project supervisor and one of his PhD students in 2016. Getting allocated this project meant the field (action recognition) had been chosen but the exact project was yet to be defined. Hence, the first step was to read this paper and the latest action recognition research papers that had been published by the Imperial Computer Vision and Learning Laboratory (ICVL) [18, 38, 29, 7, 16].

First, decision forests-based methods have been very successful and popular in many computer vision tasks because of their efficiency both in training and testing, their inherently multi-class handling ability and their capacity to handle overfitting. Hence, [16, 7] proposes a new discriminative framework based on Hough forest that enables recognition of sequential data in the form of spatio-temporal trajectories. They also introduce the concept of 'transition' that enforces the temporal coherence of estimations and enhances the discrimination between action classes. Similarly, [18] proposes a new forest based classifier that learns both static poses and transitions in a discriminative way, with a training procedure that helps to capture temporal dynamics in a state-of-the-art way. It introduces temporal relationships while growing the trees and also uses them in inference to obtain more robust frame-wise predictions.

Then, other papers are about guiding the learning process with additional knowledge in order to have better performance at the testing stage. [38] provides an end-to-end solution to action recognition from raw depth sequence. It proposes a privileged information-based recurrent neural network that exploits additional knowledge to obtain a better estimate of network parameters. RNN is indeed naturally suited for modeling temporal dynamics of human actions as it can be used to model joint probability distribution over sequence, especially in case of a long short-term memory (LSTM) which is capable of modeling long-term contextual information of complex sequential data. In a similar fashion, [29] introduces the concept of Kinematic-layout-aware random forest. They integrate kinematic-layout information (relations between scene layouts and body kinematics) in the split criteria of random forests to guide the learning process. This is done by measuring the usefulness of this type of information and closing the gap between two distributions obtained by the kinematic-layout and the appearance, if the kinematic-layout is useful. It essentially exploits

the scene layout and skeleton information in the learning process, thereby capturing more geometry that provides greater discriminant power in action classification.

This reading gave a first peak into action recognition and an understanding of the research at Imperial in this field.

2.3.2 On related fields

I then read papers on related fields that Dr. Kim's PhDs, such as Guillermo Garcia-Hernando and Daphne Antotsiou, advised me to look into. Indeed, using ideas from other fields in one's own work can help finding research ideas never tried before that then have a potential for innovation.

Hand pose estimation

Discussing with Guillermo, he especially outlined how hand pose estimation was a limit to how much action recognition techniques could improve, as estimated hand pose are often the input to an action recognition algorithm. Indeed, the hand pose can be difficult to estimate when there is an object in the user's hand as it obstructs the view of the camera. Hence, he invited me to read [30] that proposes to utilize information about human actions to improve pose estimation in videos. He also brought forward the idea of an action recognition algorithm that would be trained directly on hands holding the objects, instead of difficulty extracting the hand pose in such a scenario and then feeding the recognition algorithm with uncertain hand pose estimations. On this matter, [37] presents a novel framework for simultaneous detection of click action and estimation of occluded fingertip positions from egocentric viewed single-depth image sequences.

Lastly, [17] focuses on studying the use of 3D hand poses to recognize first-person hand actions interacting with 3D objects.Towards this goal, it collected RGB-D video sequences and is the first benchmark for RGB-D hand action sequences with 3D hand poses. This paper especially caught my attention as it directly shows that having hand-object images in the training set is crucial to train state-of-the-art hand pose estimators, likely due to the fact that occlusions and object shapes need to be seen by the estimator beforehand. Discussing with Dr. Kim, it came to my attention it would be interesting if I were to produce some work using this new dataset.

Transfer learning

While Guillermo shared his interest on hand pose estimation, Daphne did the same about her research interest in transfer learning. She is looking forward to use it to train robots to grab objects. She mentioned the original paper [12] about one-shot learning (from December 2017) that really amazed and intrigued me. One-shot learning is the name of a simple model that maps a single successful demonstration of a task to an effective policy that solves said task in a new situation. It uses soft attention to generalize to tasks unseen in the training data. The first demonstration of the effectiveness of this approach was on a family of block stacking tasks. However one-shot learning was initially designed for action reproduction [22, 25] rather than recognition. Hence I am still searching for an occasion to apply it in my project. One paper [14] gives an example of this by developing a one-shot real-time learning and recognition system for 3D actions. The proposed method relies on descriptors based on 3D Histogram of Flow and on Global Histogram of Oriented Gradient. This could be an eventual future direction to the project.

2.3.3 On grouping

After the above reading, I then came back to my supervisor who gave me a line to guide my research. Hence, he proposed to start by benchmarking with a few baselines the new dataset presented in [17] by grouping the actions by categories. This had not been tried before on this dataset and was building on previous work from PhDs. The goal was to see if grouping actions could improve the performance. Indeed, actions similar to each other are still likely to be mixed up during action recognition, resulting in a decrease in action recognition performance. Looking into previous work in grouping actions, I came across a paper [19] in the field of activity recognition. It proposes a lightweight method to improve the recognition performance by reevaluating recognized activities and grouping similar activities to a new activity. It evaluates the method on an activity-set with seven activities, containing three step-down activities that were similar to each other and, therefore, likely to be mixed up during recognition. The remaining four activity were walking activities. It shows an improvement of the recognition performance of up to 6.6%. Hence this paper shows this is a possible direction of research as grouping can improve performance by helping to distinguish between actions that are usually mixed up and bring performance down.

Similarly, trajectory group selection has been tried [27]. Indeed, for most actions, only a few local motion features (e.g., critical movement of hand, arm,leg etc.) are responsible for the action label. Therefore, highlighting the local features which are associated with important motion parts will lead to a more discriminative action representation. It introduces a motion part regularization framework to mine for discriminative groups of dense trajectories which form important motion parts. The proposed framework achieves the state-of-the-art performances on several action recognition benchmarks.

Lastly, another recent (2017) paper interestingly creates a bridge between multi-task learning and grouping [24]. It proposes a hierarchical clustering multi-task learning method for joint human action grouping and recognition. The results show that their technique can overcome the difficulty in heuristic action grouping simply based on human knowledge. It also avoids the possible inconsistency between the subjective action grouping depending on human knowledge and objective action grouping based on the feature subspace distributions of multiple actions. To be clear, multi-task learning co-learns all tasks simultaneously and optimizes the learning across all of the tasks through some shared knowledge, while transfer learning aims to learn well on a new task by transferring some shared knowledge from previously learned tasks [3].

2.4 Final report literature review

Following the interim report, those are the main research papers that influenced my thoughts on my work and the action recognition field. Firstly, [28, 6] gave an example of how to use neural networks in a tree configuration while highlighting many of the challenges involved. Then, [36] showed how multi-label representations (using multiple-verbs to describe an action) is under-explored in action classification and should be a natural evolution of the field. Indeed, a more explored research field such as object recognition now constantly uses multi-label representations. Lastly, [35] delivers an example on how to use covariance between skeleton joints motions to extract features and use them to train a RNN ("Co-occurrence Feature Learning for Skeleton based Action Recognition using Regularized Deep LSTM Networks"). Furthermore, [8] gave an overview on the effectiveness of learning both hierarchical and temporal representation with RNNs.

Chapter 3

Manual grouping

3.1 Introduction

Manual grouping consists in subjectively creating groups of actions by associating classes following a similar pattern in their hand pose sequences. In this chapter, different ways of grouping are experimented. This allows to discover the challenges involved in manual grouping and brings insight about the use of patterns in 3D hand pose sequences.

3.2 Creating a baseline

The first step was to establish a baseline to use for comparison. The $LSTM-1$ [17] Recurrent Neural Network (RNN) architecture used in the ICVL research paper was chosen for its simplicity, yet strong performance and suitability for temporal sequences.

3.2.1 Parameters

The training parameters are summarized in table 3.1 and were optimized to get maximal accuracy using this architecture and dataset. Adding layers could have improved accuracy further but it was agreed to keep the model simple and trainable in a short time by stopping at one hidden layer.

Table 3.1: Baseline network training parameters

Parameters	Learning rate	Epochs number	Batch size	Dropout probability	Hidden units number	Layers number	Optimizer type
Value	0.003	200	20	0.5	100	1	*Adam*

No mean pooling is performed to infer the output. The prediction output of each input sequence is simply retrieved from the last dynamically computed output, which is the output of the RNN cell that received the last bit of the input sequence. Figure 3.1 illustrates this for clarity.

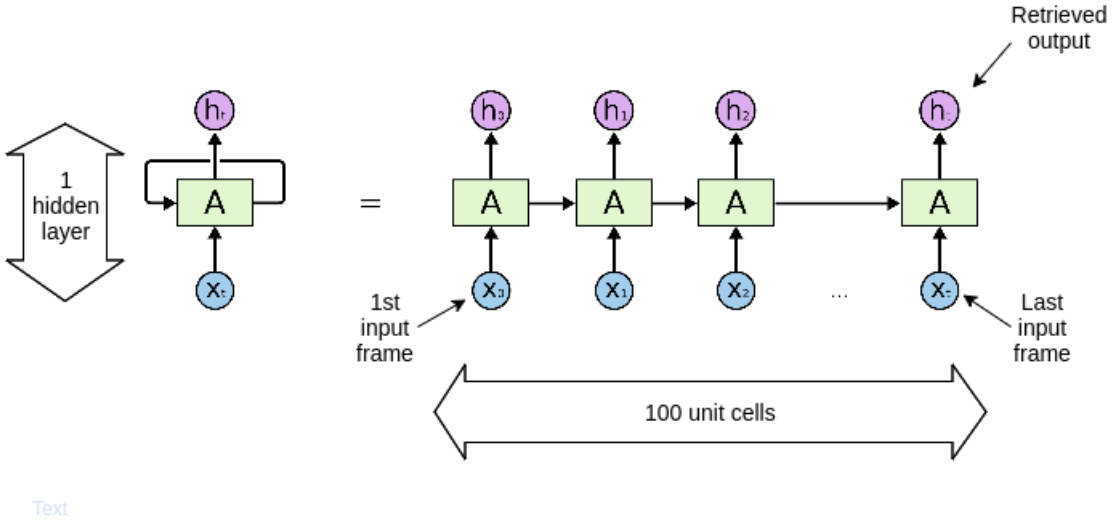


Figure 3.1: Diagram of the baseline RNN used, with an unrolled version of the network (right).

The dataset parameters are in table 3.2. The input to the RNN are the (x, y, z) coordinates of the 21 hand joints, meaning there is 63 input neurons. The length of the sequence was set high enough to include fully every sequence and pad them with zeros until they reached the *sequence padding size* (see 3.2). Then, the number of output classes, set to the number of actions in the dataset, corresponds to the number of output neurons. For later experiments with RNN trained on groups, the number of output classes will be the number of groups.
Concerning the split between train and validation sets, it is done within the training set. The data is always randomly shuffled before each split while making sure to keep the same proportion of classes in the two resulting sets. Furthermore, during training, each batch is made of a randomized data sample from the train set.

Table 3.2: Baseline dataset parameters

Parameters	Input feature size	Sequence padding size	Number output classes	Training:Testing split	Train:Validation split
Value	$21 * 3$	300	45	1 : 1	4 : 1

For each of the other experiments, those parameters are kept constant in order to reduce the time involved in spanning the parameter space searching for optimal values. In addition, some trials were made to see if it was key to tune those parameters when the number of class changes. Results showed they did not influence the accuracy by more or less than 2%. This is likely due to the fact the dataset stays the same throughout the experiments and the number of classes does not change critically, as of an order of magnitude for example. Lastly, every experiment is run five times and its accuracy result is the mean of all those runs. The confusion matrix is taken from the experiment that had the closest value to the mean.

3.2.2 Results

The baseline prediction output on the 45 action classes is available in figure 3.2. It achieves a 71.00% recognition accuracy.

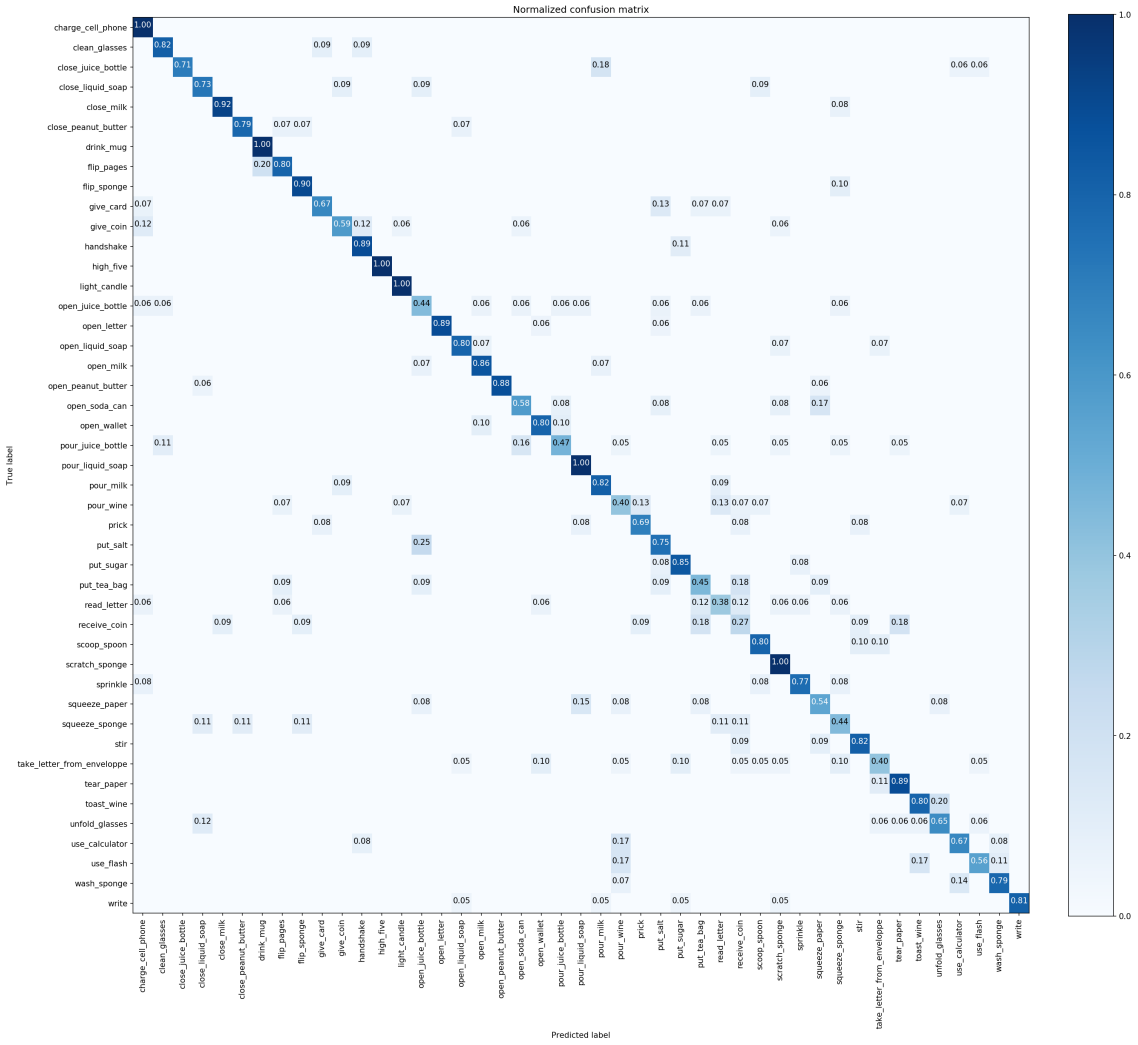


Figure 3.2: Visualization of a 45 action classification on the RNN baseline using a confusion matrix

The results are quite scattered as many input samples from one class are mistakenly identified as belonging to another class. However,by looking at input samples from successful and failed classification cases, some insight is already gained.

Figure 3.6 shows hand pose frames taken from the middle of sequences from three successful classes. During testing, the first two classes achieve a 100% classification success on all their input samples while the third one is at 81% but has double the number of samples to classify. Observing each of those sequences shows a clear pattern between them: the hand grasp does not move throughout the motions. For example, during the *high five*, the hand structure stays the same and only the elbow extending pushes the hand forward. During the *write* action, the hand stays firmly fixed around the pen and only the forearm and wrist produce the motion. Similarly, during the *charge cell phone* action, the hand holds the charger plug firmly like the pen and the elbow pushes the hand forward into the phone plug. This situation also applies to many other actions that perform well on this baseline. They often incorporate very little movement, simple one way motions or recurring short motions, and involve no rotation.

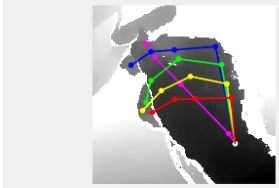


Figure 3.3: charge cell phone

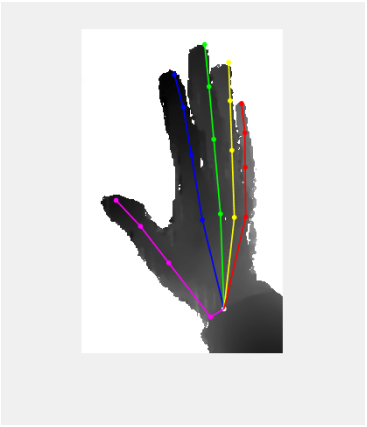


Figure 3.4: high five

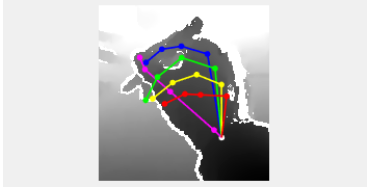


Figure 3.5: write with pen

Figure 3.6: 3 successful cases of action classification

On another hand, figure 3.10 shows three classes with low classification performance (respectively 38%, 27%, 44%). Their common point is that the hand grasp varies greatly throughout their sequences. For example, during the *receive coin* action, the hand unfolds and stretches out to receive the coin. During the *squeeze sponge* action, the hand successively tightens and expands around the sponge which modifies its joint structure considerably. Similarly with action *read letter*, the hand extends to reach the paper, tightens to grasp it, and rotates to put the paper in a vertical position.

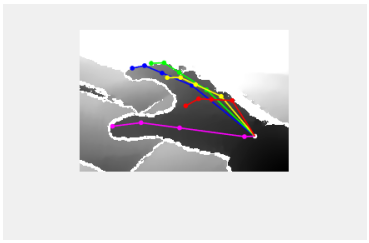


Figure 3.7: read letter

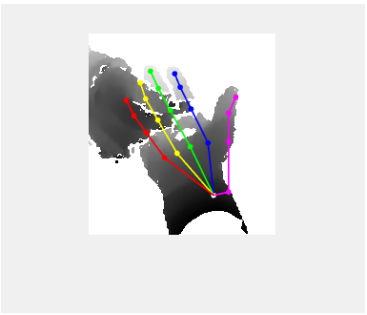


Figure 3.8: receive coin

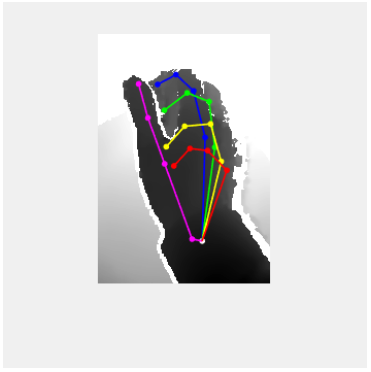


Figure 3.9: squeeze sponge

Figure 3.10: 3 failed cases of action classification

Hence, the RNN seems to perform well on sequences with constant joint structure (or grasp) where the motion is linear (no rotation). This makes sense as training an RNN on those kind of sequences is like training a CNN on similar images that have a linear transformation between each other. Hence, the network associates well the image of this specific hand grasp with its class. But for classes where this grasp varies, this process can not happen as the network is unable to generalize by looking exclusively at the hand structure. This is likely to be reinforced by the fact the last output of the RNN is taken as the global output, meaning its last input frame influences as much as its past input frames (put simply). Thus, for constant grasp actions, the joint structure information from the last input frame reflects well the overall sequence. Whereas for failed classes,

the last hand grasp input might be very different from the past ones and the RNN ends up with conflicting information on the hand structure, at the output.

Lastly, without looking at the exact nature of the data itself, two observations can be made. First, classes with more training data available tend to perform better during testing, which is a well-known and documented aspect of neural networks. Second, classes with greater variability among their training input samples perform worse, as the network ends up having less material to generalize the action since it covers a wider range of possible movements. However this variability may lead to a better generalization in real life testing as different people can execute the same action with a degree of diversity in the movement. To sum up, the quantity and quality of the data plays a role as important as the architecture in the neural network's performance.

3.3 Creating groups

Using insights gained from the 45 actions baseline RNN, the next step was to create groups by aggregating classes in a sensible way. A group implies a common pattern between the classes inside this group. Hence, watching over each action sequence, two patterns seemed to come to light. The first is a static pattern concerning similar objects. Indeed, for different actions using the same object, the hand grasp on the object is most of the time fixed although the hand is moving around. The second pattern is a dynamic pattern. For similar actions on different objects, the motion of the hand in space is similar although its grasp on the object is different. Hence, it made sense to group the 45 actions in two ways, one through objects and one through motions. This allowed to separate and evaluate group recognition performance on hand structure (how each of the 21 joints is positioned relative to the others) and hand motion (how the whole 21 joints move together into space). This grouping also links well with observations made on the baseline where both constant grasp and/or constant motion sequences achieved high recognition accuracies.

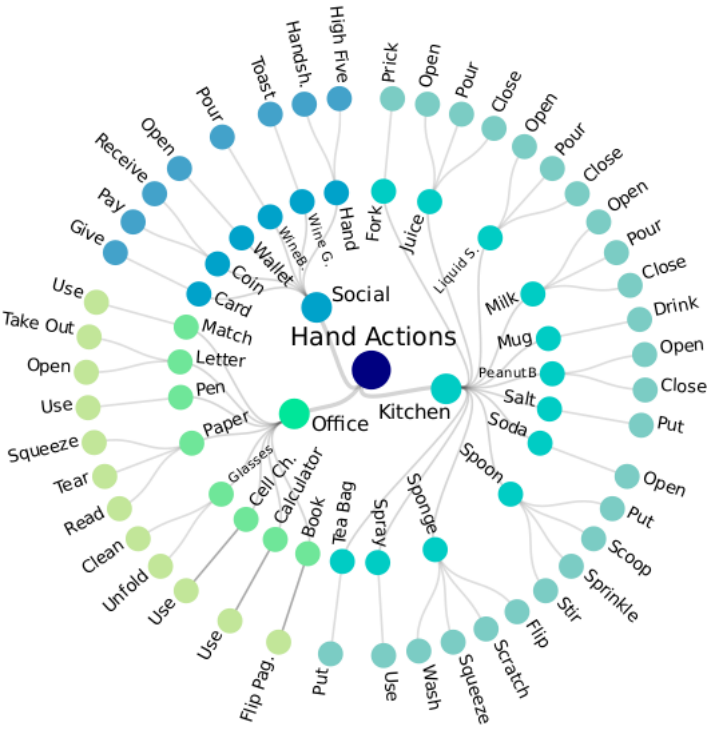


Figure 3.11: Original 45 action classes from the ICVL dataset. Image borrowed from [17].

3.3.1 Static hand object grasp

The grouping is done according to common object grasp and is available in figure 3.12. Groups, or 'super-classes', include from 1 to 4 actions. The grouping is imperfect as it is done subjectively and is unbalanced with many 1 action groups. This makes comparing group recognition results less fair, but is inevitable due to the way the actions were decided to be grouped. Indeed, most of the 1 action groups have specific hand grasps that can be difficultly linked to another class.

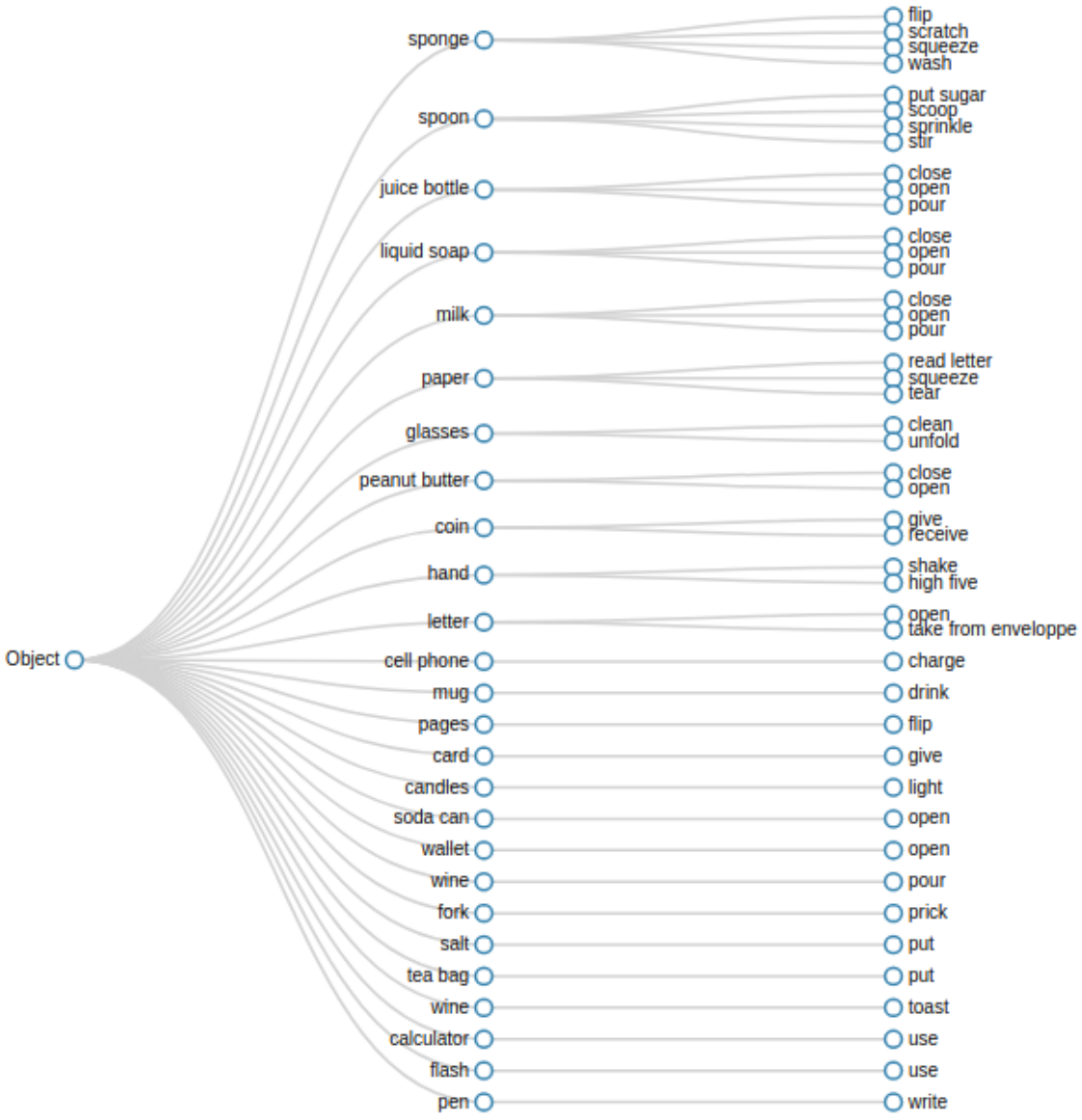


Figure 3.12: 26 Object group organization of the 45 action classes

3.3.2 Dynamic hand motion

The grouping is done according to similar motions and is available in figure 3.13. Super-classes include from 1 to 7 actions. This grouping was less straightforward to build than the previous as each action's motion is quite different and observing similarities between them is harder. Hence the super-classes are more coarse. For example, the class *meet* was built from motions where two

hands meet and stretch out (to varying degrees) during the movement, as it happens during a handshake or when receiving a coin. The class *use* is made of actions where the index always plays an advanced role in space and moves roughly in linear motions, as when pressing the buttons of a calculator or pushing a spray's control stick. The class *takeout* refers to actions where both the index and thumb are holding a piece of the object and moving it in space while the other fingers rest in the hand (unfolding glasses, taking out a letter, reading a letter). Similarly, the class *prick* includes actions where a piece of cutlery (fork or spoon) is held and inserted into a recipient to grab a quantity inside it. The rest of the actions are self-explanatory through their super-class names or are 1 action groups.

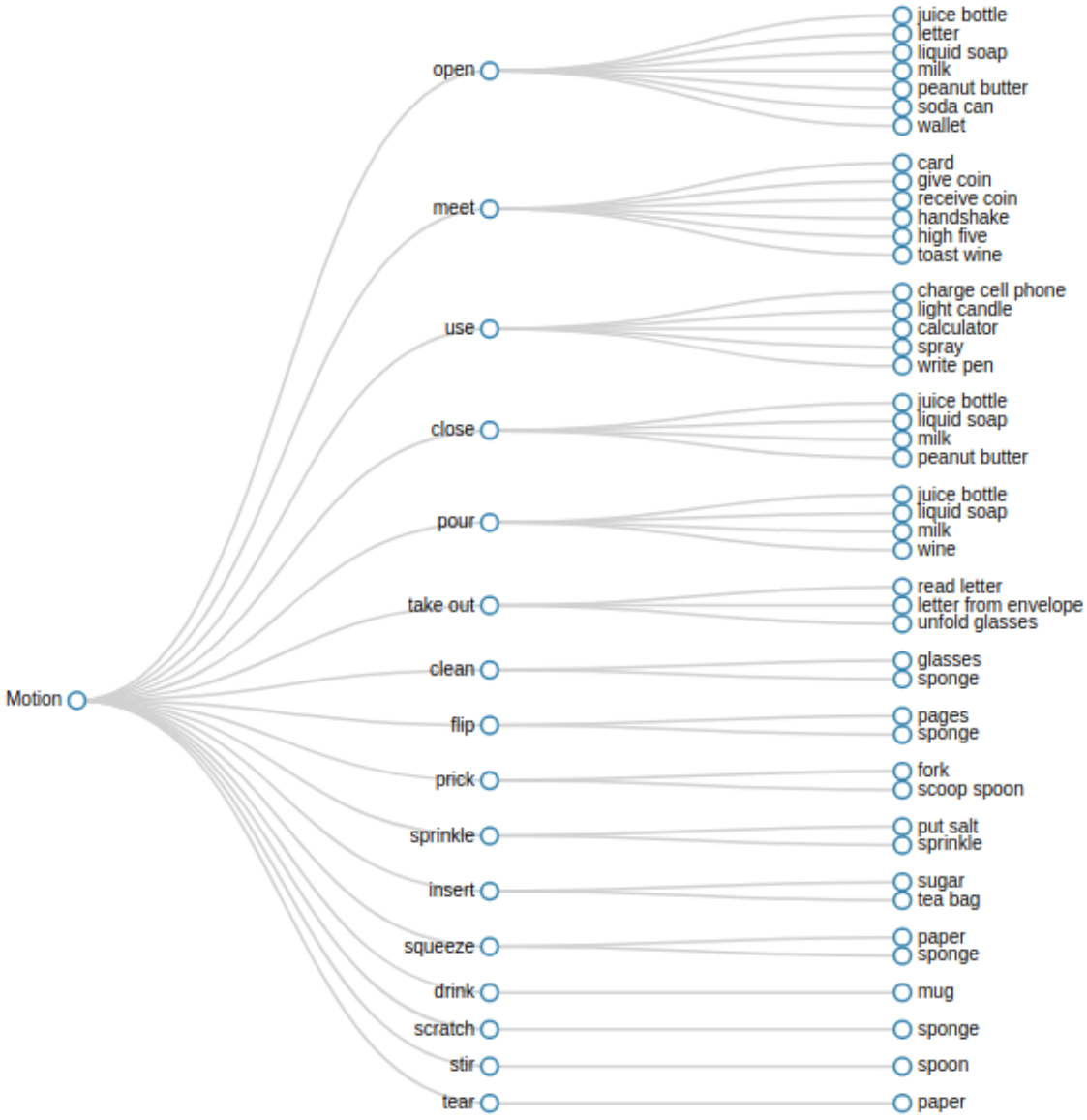


Figure 3.13: 16 Motion group organization of the 45 action classes

3.4 Testing groups

In this section, the object and motion group recognition performances are tested and compared.

3.4.1 On the baseline

First of all, the performance on the baseline is investigated. The Baseline RNN is trained to have a 45 action classes output and the group recognition accuracy is calculated *after* recognition of the 45 classes as detailed in figure 3.14.

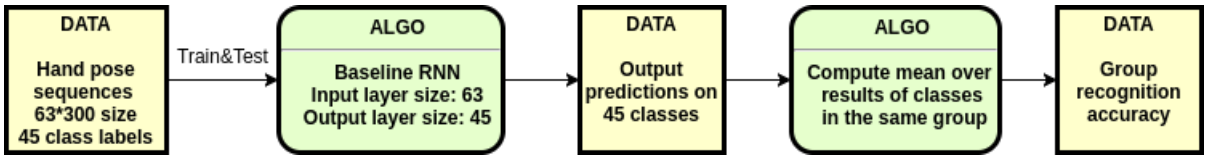


Figure 3.14: Pipeline of the group recognition accuracy experiment using the baseline RNN

One can see the motion accuracy is slightly higher but those results only make sense when compared to the later ones, as this is a simple reorganization of the 45 actions prediction results that will serve as a comparison baseline.

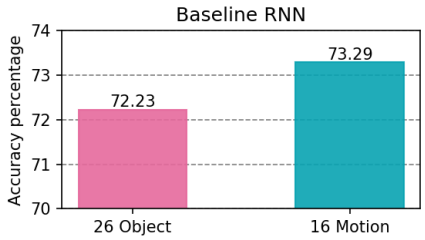


Figure 3.15: Group recognition accuracy using the baseline RNN

3.4.2 On trained recurrent neural networks

To evaluate group recognition performance, an RNN with the same network parameters as the baseline is trained on the classes of each group. Its accuracy is directly computed from its output as shown in figure 3.16. This experiment allows to see how much the RNN can improve its group recognition performance through leveraging its training on similar hand grasp or motion patterns.

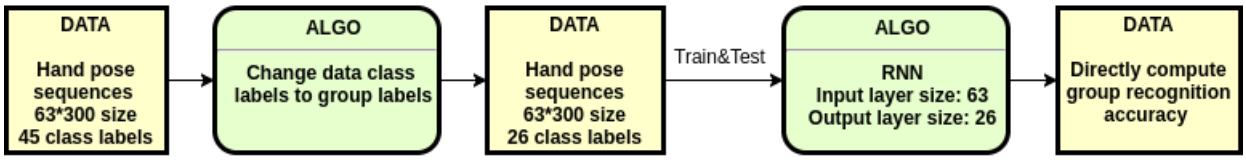


Figure 3.16: Pipeline of the group recognition accuracy experiment using the trained RNN. The 26 Object grouping is used as an example here.

Looking at the accuracy results in figure 3.17 shows the object grouping performs better than the motion grouping. This tends to indicate similar hand grasps are more easily identified than similar motions by the RNN during training.

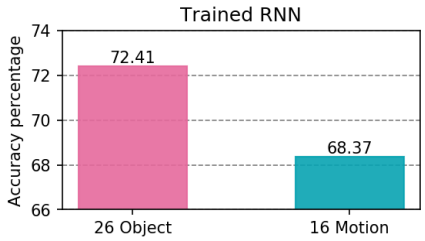


Figure 3.17: Group recognition accuracy using RNN trained on the groups

Looking into the object confusion matrix in figure 3.18 brings more insight to those observations. First of all, 3 and 4 actions group perform quite well relative to the average (sponge 80%, spoon 74%, dish soap 88%, milk 73%), while most 1 action groups perform below average (wine 60%, fork 53%, salt 62%, teabag 45%). This is likely due to the lack of training data as those 1 action groups train on 6 samples in average while other 1 action groups with double the number of training samples perform much better (wineglass 92%, calculator 92%, flash 88%, spray 88%).

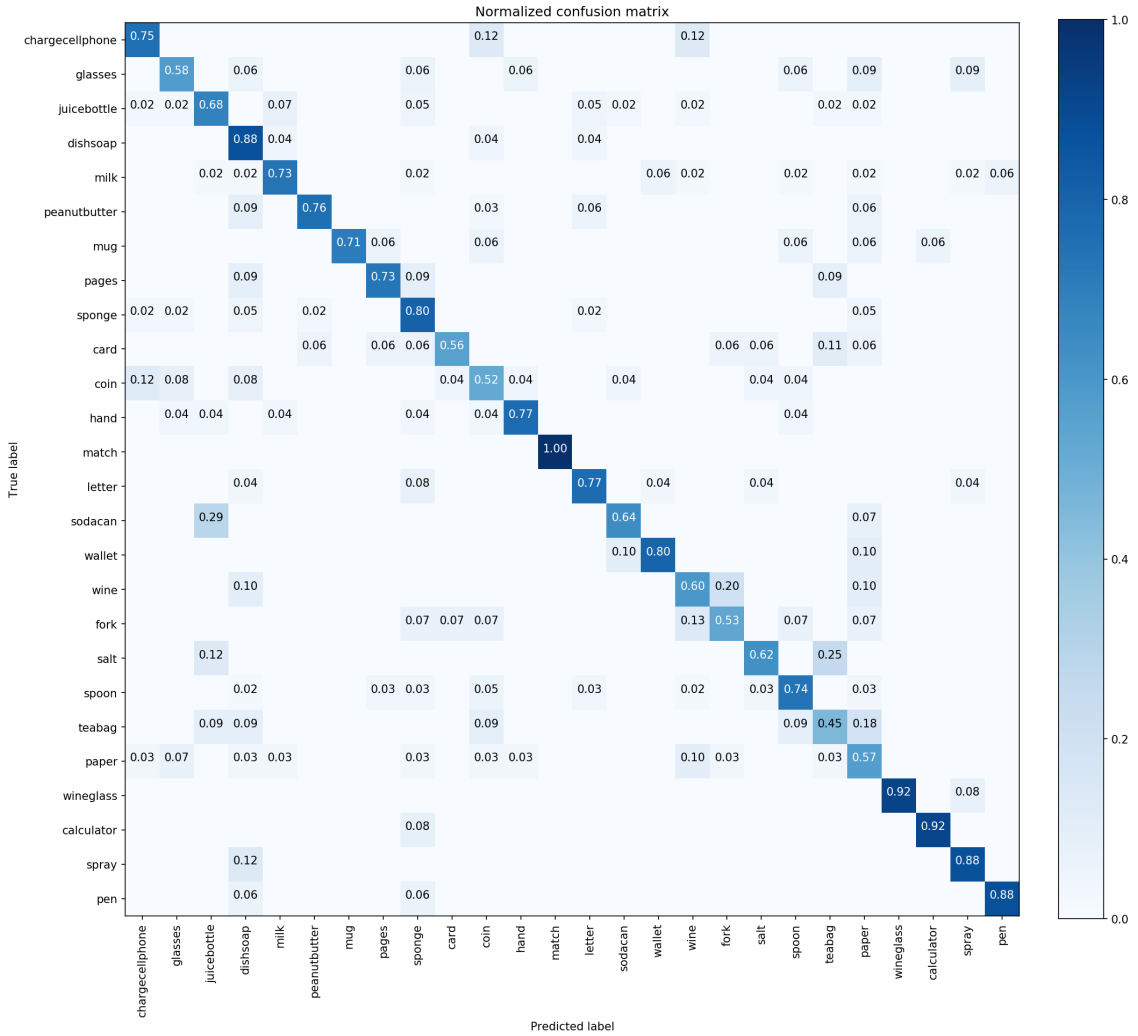


Figure 3.18: Visualization of the 26 Object group classification on the trained RNN using a confusion matrix

The *paper* group is the only large group performing poorly (57%). It also appears to be attracting a lot of misclassification (see the *paper* column).Looking at its samples in figure 3.19, it seems they span a large array of hand grasps due to varying ways in which the paper is held throughout the actions. Thus, this can confuse the classifier and steal some predictions from other classes. Furthermore, the situation makes sense as the shape of a paper is very thin and does not ideally fit in the palm of a hand, unlike other objects do (glass, bottle, jar).

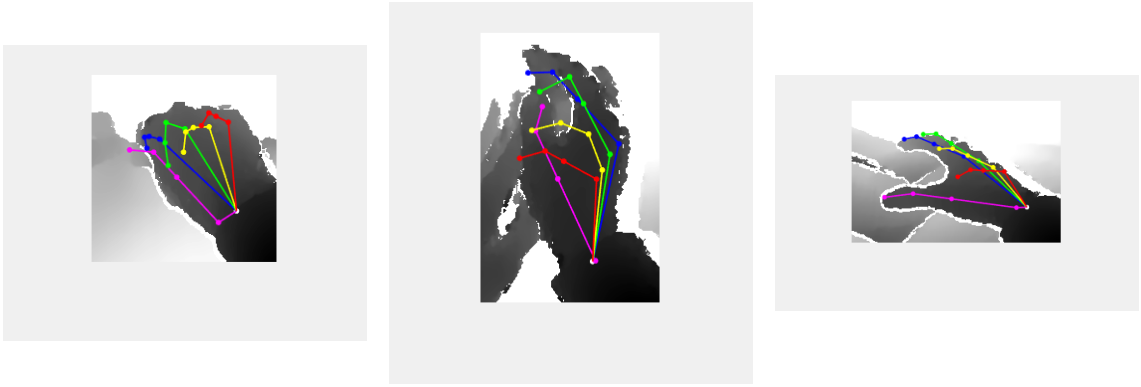


Figure 3.19: Variety of hand grasps from the *paper* group

Lastly, a part of the 2 action groups have a low performance (below 60%: glasses , card, coin) while another part (above 75%: letter, hand) performs better. Looking at the data, it is confirmed the latter have a better match between the hand grasp patterns.

Next, the motion confusion matrix in figure 3.20 shows large groups (4 to 7 actions) perform very well (open 73%, meet 67%, use 83%, close 83%, clean 85%, pour 76%). Their high performance can be explained through the *clean* group example where its two member actions have a similar rotation motion and hand grasp on the object, as shown in figure 3.23.

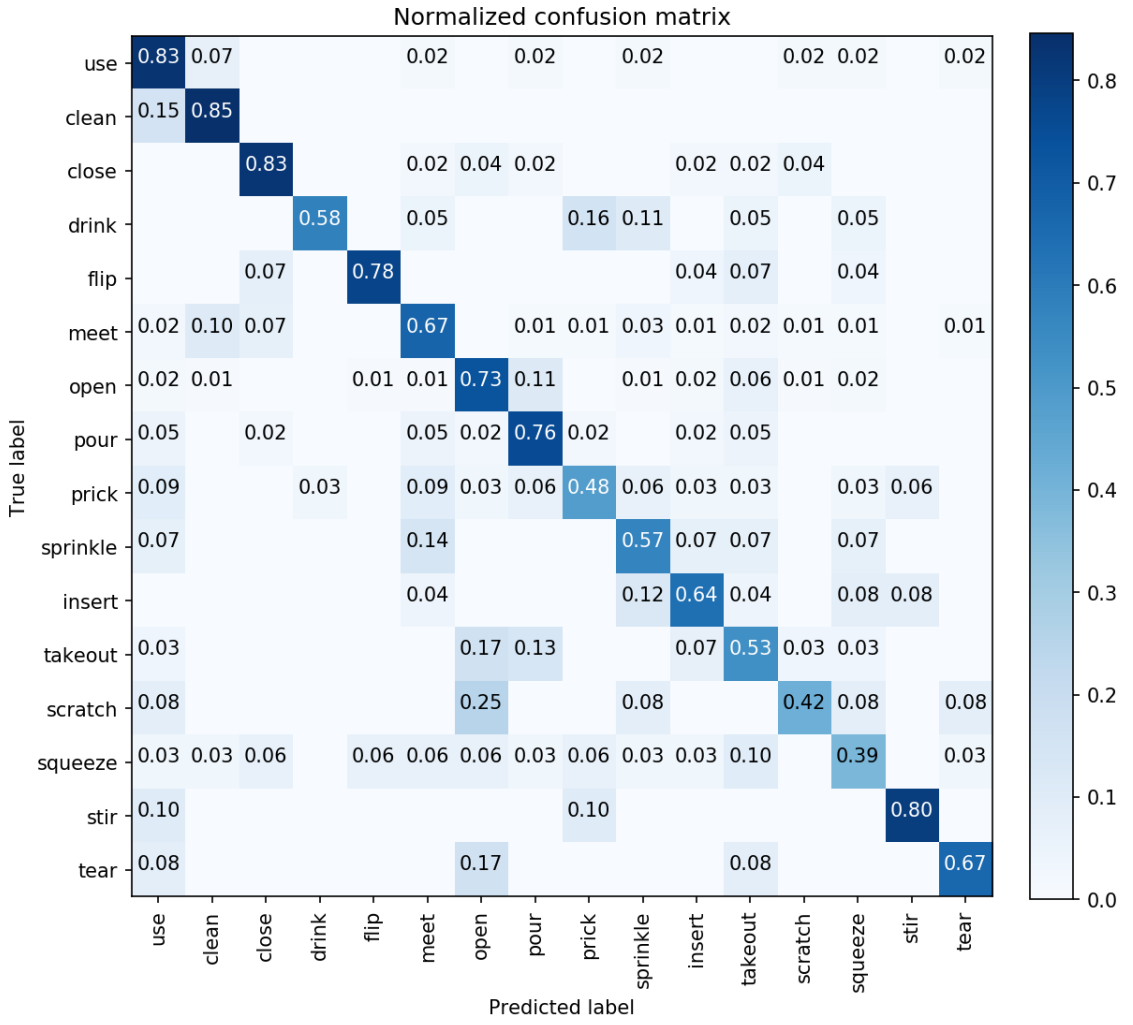


Figure 3.20: Visualization of the 16 Motion group classification on the trained RNN using a confusion matrix

The *use* group 83% performance is particularly impressive as the grouping was a daring one with actions that visually seemed to have little in common. The main aspect they shared was a recurring loop of small forward and/or rotating motions, where the index was responsible for orienting the action in a direction, by being ahead of the other fingers in the 3D space. This is shown in figure 3.27 and indicates the finger placements such as the index indeed plays a central role in helping the network recognize the action. This is further supported by the results in the ICVL paper [17] that report 65% of actions can be identified solely from the index finger's joint coordinates.

Less successful motion cases (prick 48%, scratch 42%, squeeze 39%) are also investigated. 25% of *scratch* misclassification are in the *open* group. This could be explained by the fact this action is mostly a hand holding a sponge and rotating it, which can be assimilated to holding a bottle cap and rotating it to open it. The *prick* action has interesting insight: both actions (prick with work and scoop with spoon) have the same grasp on the object and visually look like they have a similar motion. But when looking at the pose visualization, it appears the hand is going up and down with the fork whereas it is rotating 90 degrees with the spoon. Hence for similar grasps, rotating and forward motions are difficult to associate together for the network. Lastly, the *squeeze* group

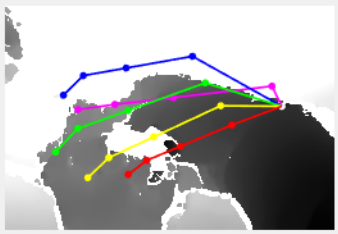


Figure 3.21: action clean glass

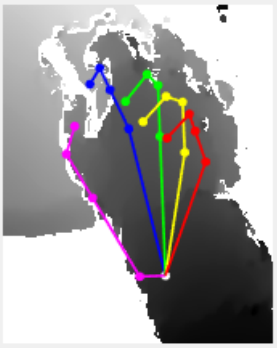


Figure 3.22: action wash with sponge

Figure 3.23: Variety of hand grasps from the *clean* group

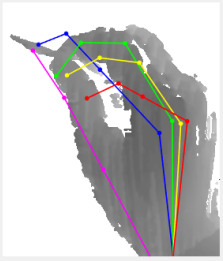


Figure 3.24: action light candle

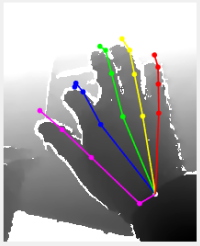


Figure 3.25: action use calculator

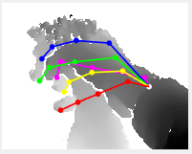


Figure 3.26: action use spray

Figure 3.27: Variety of hand grasps from the *use* group

results are surprising as both actions (squeeze paper and squeeze sponge) execute the same motion. The only difference in the sequence is that the *squeeze paper* action manipulates the object before crushing it whereas the crush motion happens right away with the sponge. Generally, those 3 actions have very little motion in their sequences, meaning it becomes hard for the network to extract a motion pattern from them. Hence, the RNN can only extract a similarity in the hand grasp, which is not what the actions have in common inside similar motion groups. This explains why the testing samples are misclassified in almost all other groups, as if the classification had became random because the network is unable to generalize on different motion samples having the same label.

Finally, compared to the object group, the results are more clear cut. A third of groups perform very highly (above 75%) while another third that are less ideal motion matches, perform very poorly (below 55%). This could mean motion is a more effective form of grouping than grasp when patterns are matched properly, and worse otherwise.

3.4.3 On the number of actions

Due to the previous results, investigating results per number of actions in a group became worthy of interest. Figure 3.28 serves this purpose and demonstrates clearly how 1 action groups bring the performance down compared to multi-action groups for each grouping. This is likely due to the fact 1 action groups have less training data to provide to the network than multi-action groups. Hence the RNN is more poorly trained on those patterns and is unable to generalize as strongly as it does with the larger groups. This can lead to those 1 action group samples being misclassified into more powerfully trained large groups. Having balanced groups would indeed have been better to prevent those occurrences. However, the quantity of data available and the motivation behind the grouping methods prevent this. Figure 3.28 also shows grouping by patterns might be working better than expected since the performance of multi-action groups is above the overall accuracy that is hindered by 1 action groups. One critic could be that this is due to the smaller number of multi-action groups (less options leads to less potential misclassification) but this is unlikely as for the motion grouping, 80% of groups are multi-action and the multi-action group accuracy is still above the overall group accuracy.

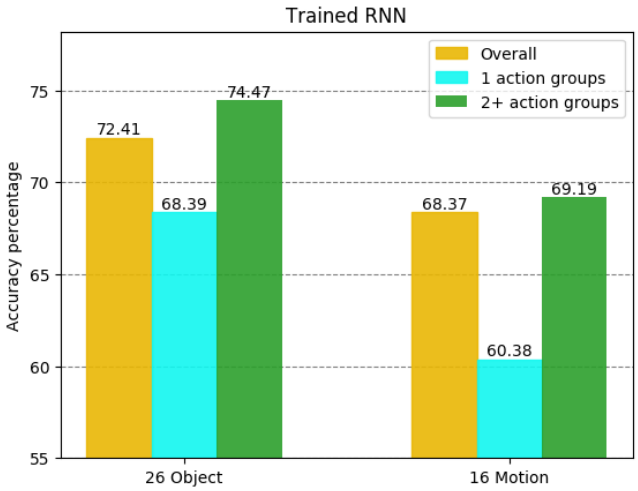


Figure 3.28: Group recognition accuracy of the trained RNN decomposed by number of actions inside a group. The 1 action groups accuracy is computed only over 1 action groups classification results (same for 2 or more action groups).

Table 3.3: Number of groups per action(s)

Decompostion	1 action	2 or more actions
26 Object	11	15
16 Motion	12	4

3.4.4 Comparison

Combined with the previous baseline experiment, the last goal is to identify whether training the RNN on differentiated actions achieves better group recognition accuracy than training it on grouped actions. The results are shown in figure 3.29 and indicate the RNN is indeed able to generalize well on hand grasps patterns (Object grouping) as it beats the baseline, but achieves this with more difficulty on hand motions (Motion grouping). The main reason for this is that

some action classes lack a discriminative motion the RNN could use to differentiate them from other actions, whereas most action classes contain a discriminative hand grasp to take advantage of.

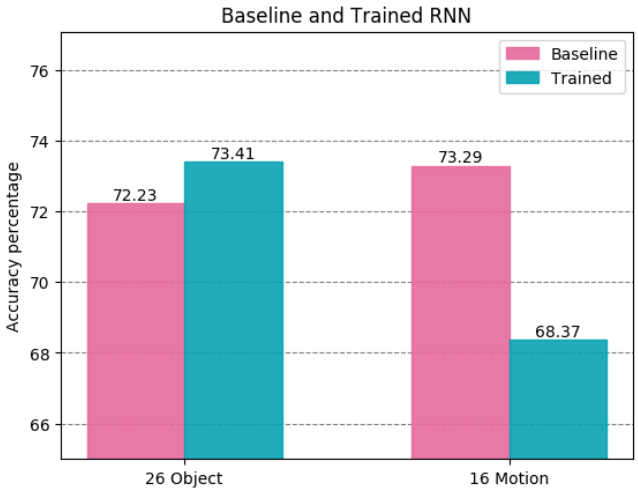


Figure 3.29: Comparison of the group recognition accuracy between the baseline and trained RNN

3.5 Trying new groups

As shown and explained previously in figure 3.28, 1 action groups reduce performance considerably compared to multi-action groups. One attempted solution was to regroup all the heterogeneous single actions in one group so the RNN would perhaps identify this group as the one to choose when no clear (or mixed) group patterns are identified. Hence, the two new group configurations available in figure 3.30 and 3.31 were formed.

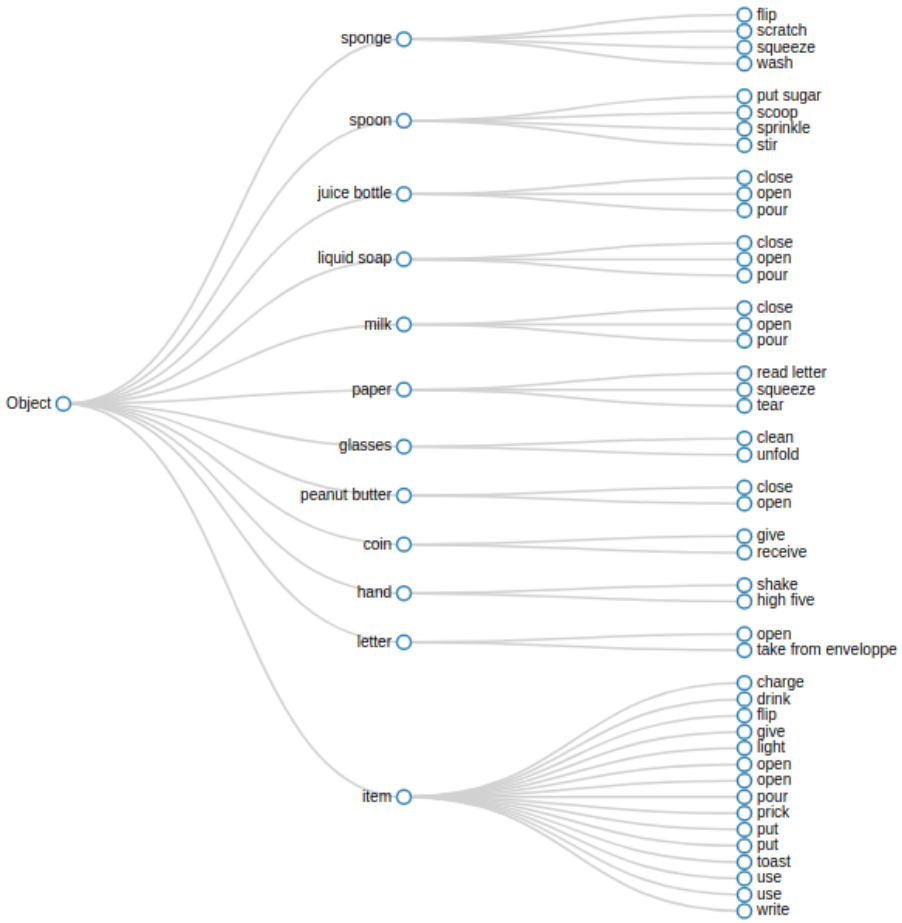


Figure 3.30: 12 Object group organization with all 1 action groups gathered in the *item* group

The results are available in figure 3.32. They show the new Motion grouping indeed benefits from the merging of 1 action groups as it performs better than the old Motion grouping, while the baseline stays identical (which is not an experiment error). However, the performance of the new Object grouping is worse as it decreases compared to the old grouping, while the baseline increases. Knowing 60% of the old Object groups were 1 action groups (versus 20% for the old Motion groups), this might mean the new 'mixed' actions group deteriorates the network's performance because it is too large and unbalanced compared to the other groups. Indeed, more than half of the network's training is now spent on learning to classify unrelated sequences into a 'miscellaneous' group. It ends up being like injecting random noise in the network's training. This is confirmed by the 12 Object grouping confusion matrix (appendix figure A.1) which shows the new large *item* group attracts misclassification from all other groups (vertical reading) and is misclassified in all other groups (horizontal reading), thereby underlying the random nature. Hence, keeping the miscellaneous group size at the same order of magnitude as the other groups is necessary to make it a minimum effective and not counter-productive.

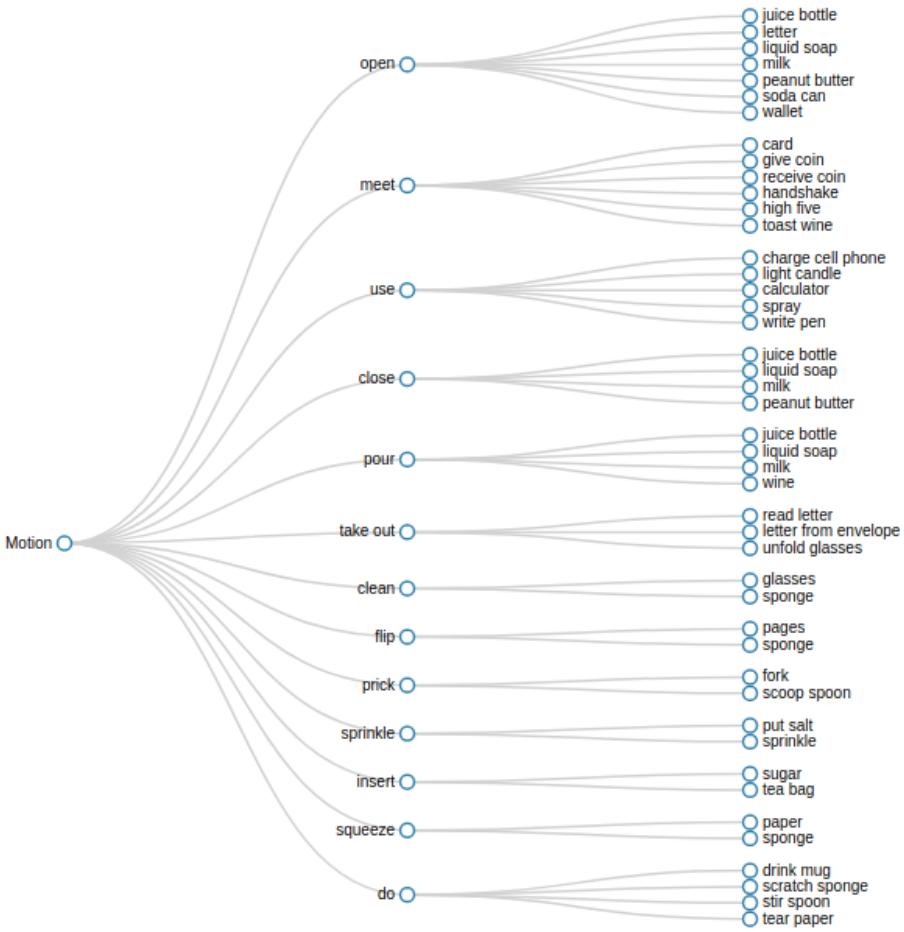


Figure 3.31: 13 Motion group organization with all 1 action groups gathered in the *do* group

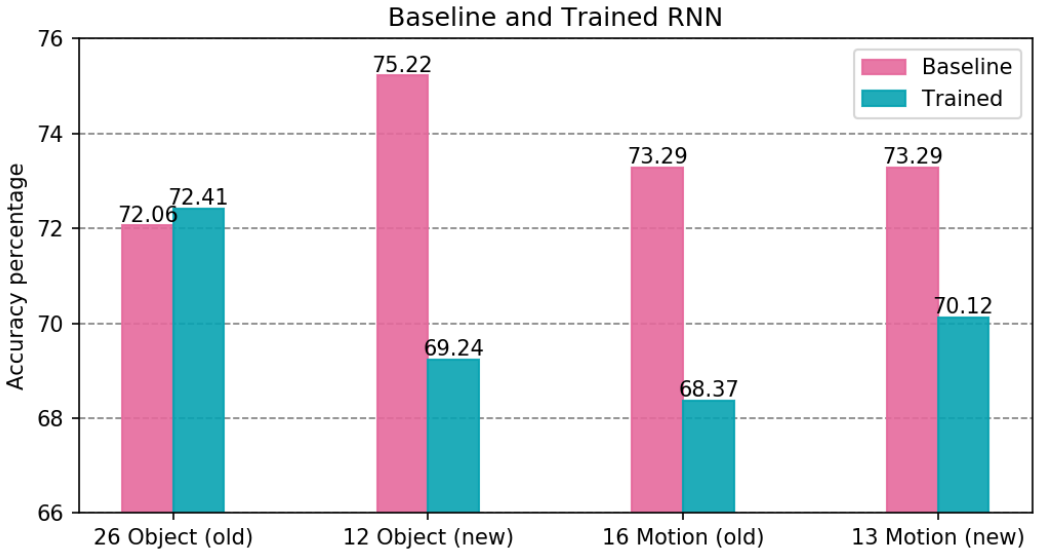


Figure 3.32: Comparison of the group recognition accuracy between old and new groupings

3.6 Instance recognition

One of the final aim of this project is to improve individual action recognition through using group action recognition. Thus, after classifying each test sequences inside a group with the trained RNN used in the previous section, the following experiment further classifies the sequence into one of the action classes belonging to this group. This second step is done using a RNN trained only on the group's actions. The whole process is detailed in figure 3.33. Note that subnets only act on multi-action groups so for the 26 Object and 16 Motion grouping, the sequences classified in 1 action groups are directly sent to the "aggregate prediction results" algorithmic box.

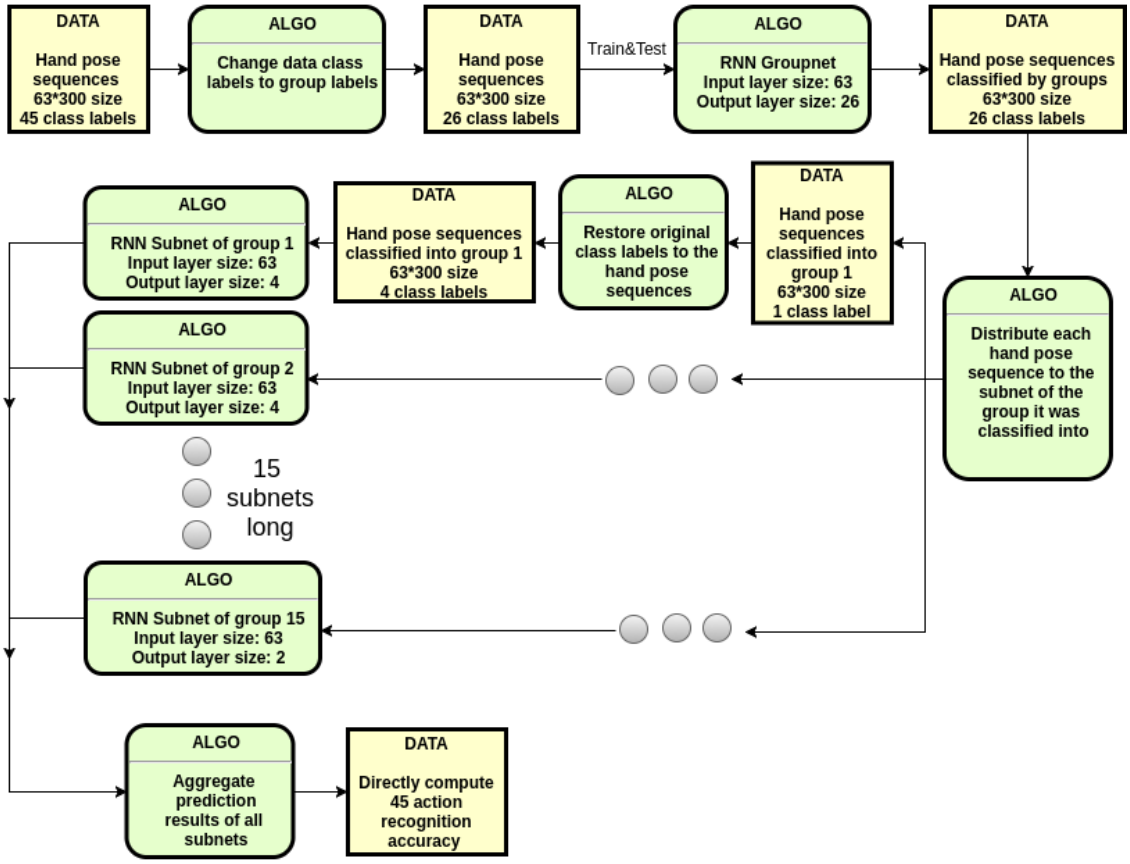


Figure 3.33: Pipeline of action recognition accuracy experiment using a treenet. The 26 Object grouping is used as an example here.

For the sake of clarity, keywords are used: the group recognition RNN is called a 'groupnet' and distributes the sequences to different 'subnets'. Subnets are RNN trained only on differentiating the actions inside a specific group. Hence, the groupnet classifies each sample inside its group or 'super-class' while the subnets classify each sample into their action class or 'fine-class'. The 'treenet' refers to the whole process and can be thought of as "$treenet = groupnet + subnet$". The three terms are defined visually in figure 3.34.

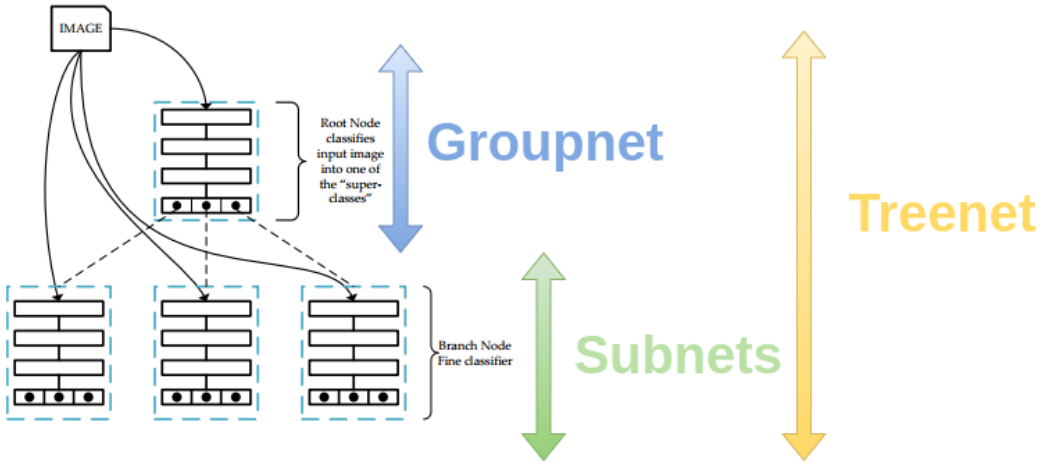


Figure 3.34: Diagram of the treenet. Image borrowed from [28].

3.6.1 Parameters

Tests were made to optimize the subnets parameters and showed they should be kept identical to the groupnet parameters (see tables 3.1 and 3.2). Tests on the number of hidden LSTM cells necessary are available in table 3.4. No trials were made over 100 units as, intuitively, the subnets should not be bigger than the groupnet since they classify a smaller number of classes (between 2 to 7 for subnets and 12 to 26 for the groupnet). The last column of the table was an attempt to tune the number of cells used to the number of classes in the subnet's group. The test accuracy was calculated by feeding each subnets with only the sequences belonging to their respective groups and aggregating the prediction results from all the subnets to compute the test accuracy. The 13 Motion grouping was used for this test.

Table 3.4: Effect of LSTM cells number on test accuracy

Number of cells	20	50	100	3 * number of action classes
Test accuracy	79.96%	80.49%	**83.30%**	81.20%

3.6.2 Results

Results in figure 3.35 show, for each grouping, the groupnet's group recognition accuracy, the subnets 45 actions recognition accuracy and the treenet 45 actions recognition accuracy. This allows to visualize multiple information at the same time. First, the subnets perform highly and quite similarly. Second, the action recognition accuracy of the treenet (the final accuracy of the "groupnet+subnet" pipeline) is generally well below the 71% 45 actions baseline seen at the beginning of this chapter 3.2. Indeed, the error accumulates along the pipeline as it usually happens in divide-and-conquer classification strategies. The groupnet is the element bringing the performance down and its improvement would bring to a better final performance. For example, the 26 Object groupnet performs better than the others (72%) and thus leads the treenet to achieve a better accuracy (65%).

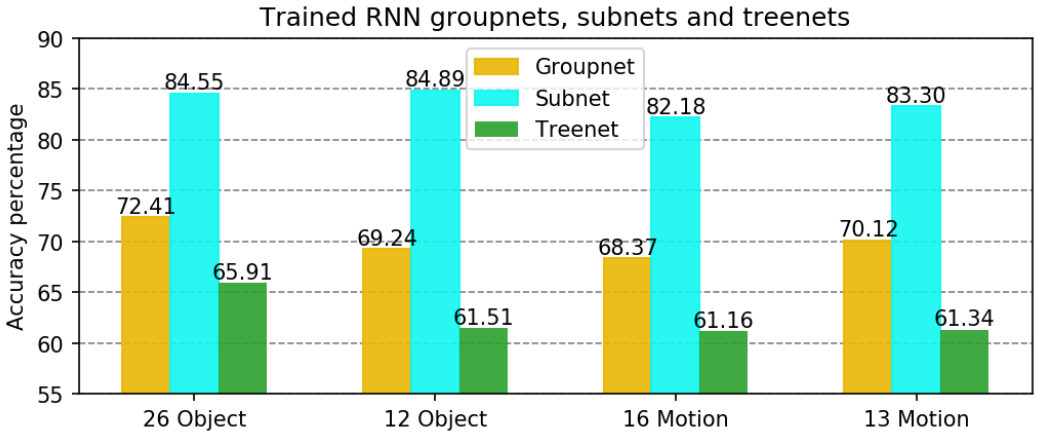


Figure 3.35: Comparison of the subnet action recognition accuracy on the 4 different groupings

3.7 Conclusion

In this chapter, groups of actions were manually created and trained to be recognized by a RNN. Results showed this method involves various challenges:

- It exhibits a loss of performance as soon as the training data from different groups of actions is unbalanced.
- For the RNN to perform well, there needs to be a recurrent discriminative pattern across all training samples of a group.
- Grouping based on a single grasp or motion pattern only works for some action classes as some of the more complex actions include evolving grasps and varying motions throughout their sequences.

However the experiments have allowed to gain insight on hand action recognition, notably that:

- Motion is a more discriminative form of grouping than grasp when the patterns between action classes of the same group are a suitable match. But, in the presence of noise between those patterns, grouping by grasps performs better in average.
- Some joints have more discriminative power than others (eg: index).
- Rotating and forward motions noise is difficult for the network to associate inside the same group.

Lastly, neural networks perform well on classifying action instances when the number of possible outcomes is reduced and even if the training set is small. Hence, it seems the true bottleneck to achieving better action recognition is the group recognition accuracy. Furthermore, this grouping method is not scalable since it is done manually. This is why next chapter will focus on grouping actions using a more systematic approach that can automatically find underlying patterns between actions and group them effectively.

Chapter 4

Automatic grouping

4.1 Introduction

In this chapter, a method to automatically group actions is attempted. The action sequences are first embedded into a single dimension vector that compresses and captures the temporal information. A clustering algorithm is then applied to the embedded sequences to create groups. Lastly, the embedding and clustering parameters are chosen in a way that optimizes the group recognition accuracy. Insight is extracted from the resulting groups and experiments.

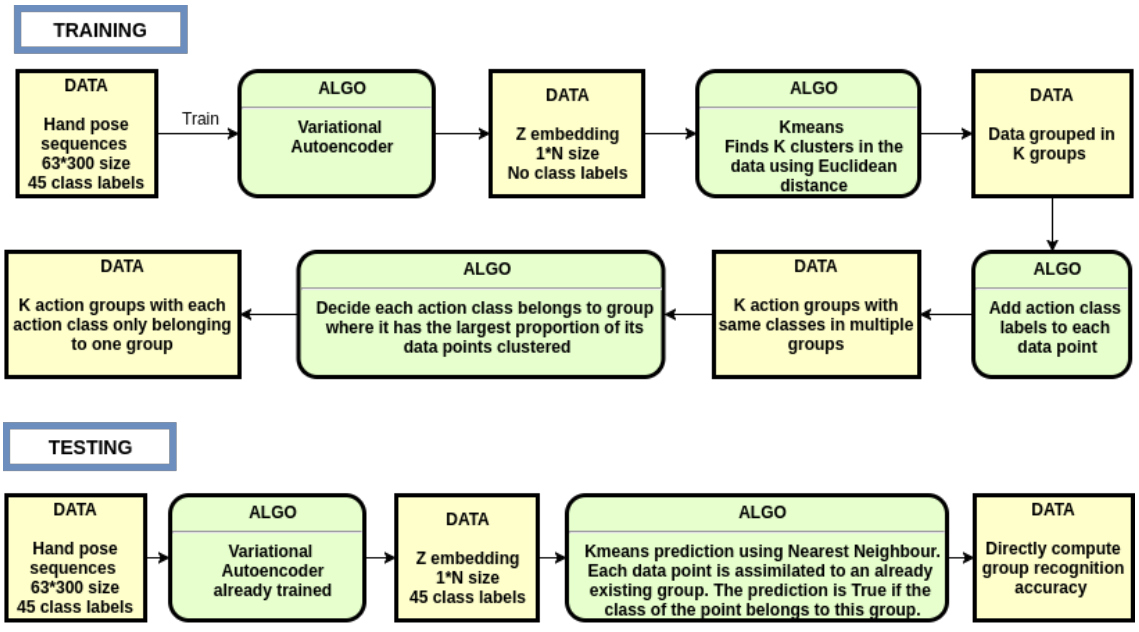


Figure 4.1: Pipeline of the automatic grouping experiment using a Variational autoencoder to embed the sequences and Kmeans to cluster them.

4.2 VAE Embedding

This first part relies on the assumption that embedding techniques can recognize complex underlying patterns invisible to the human eye and allow to perform a better grouping than what could be done by recognizing patterns manually. The embedding algorithm used is the Variational

autoencoder (VAE). Its recent popularity to embed latent information well made it an interesting choice to experiment embedding temporal information with. As [11] states, the VAE has emerged as one of the most popular approaches to unsupervised learning of complicated distributions. It is appealing as it is built on top of standard function approximators (neural networks), and can be trained with stochastic gradient descent. It has already shown promise in generating many kinds of complicated data, including handwritten digits. In general with VAE, there is no need to worry about ensuring that the latent structure exists. If such latent structure helps the model accurately reproduce (i.e. maximize the likelihood of) the training set, then the network will learn that structure at some layer. Lastly, the company 'Deepmind' has recently used VAEs to embed representations of demonstration trajectories [33]. Furthermore, applying the VAE to skeleton poses has been done previously in [32] where the VAE is used to model the possible future movements of humans in the pose space.

4.2.1 Setup and parameters

With help from [26], the VAE is first trained on the handwritten digit MNIST dataset [23]. The results in figure 4.2 show the VAE works properly from an early training stage and improves at each epochs. Indeed, the numbers get less blurry and start having a more discriminative shape while the t-SNE clusters get further away from each other. As [26] mentions, what happens is that the network is constrained to produce latent vectors having entries that follow the unit normal distribution. Then, when trying to generate data, some values are sampled from this distribution, fed to the decoder, and the decoder returns a completely new object that appear just like the objects the network has been trained with.

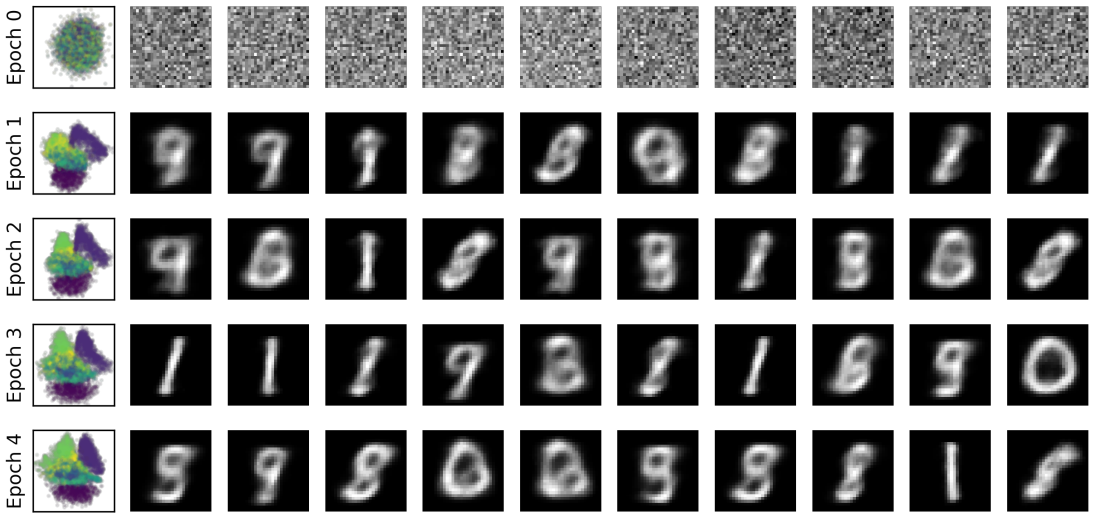


Figure 4.2: Results of using the VAE on the MNIST dataset. (Left) t-SNE 2D clusters on the 10 digit classes. (Right) VAE Decoder generating numbers from random classes.

The working VAE is then adapted to receive the action sequences as input. Each data point (or sequences) is fed as a vector of [number of joints × sequence length] dimensions and, during training, the VAE's task is to somehow capture the dependencies between those dimensions. The training parameters are available in table 4.1. The sequence length is chosen as 70 because it is long enough to capture the essential discriminative steps of every action in the dataset and it limits the time involved in running the experiment. The number of epochs is limited to 10 as the image reconstruction loss (the loss function) makes little progress after this number as shown in figure

4.3. Lastly, the embedding dimension was set to 8 as increasing it further did not bring significant improvement to the loss function. However larger embedding dimensions are tried in the clustering section to evaluate its effect on the grouping quality.

Table 4.1: VAE training parameters

Parameters	Number of joints	Sequence length	Input dimension	Epochs number	Batch size	Embedding dimension
Value	63	70	$63 * 70$	10	30	8

The image reconstruction loss used in figure 4.3 is calculating by adding together the sum of squared difference (between the original and the decoded image) and the Kullback-Leibler divergence [2]. The latter is a measure of how one probability distribution diverges from a second expected probability distribution and ensures the latent values will be sampled from a normal distribution. This combination is typically used in VAE loss functions.

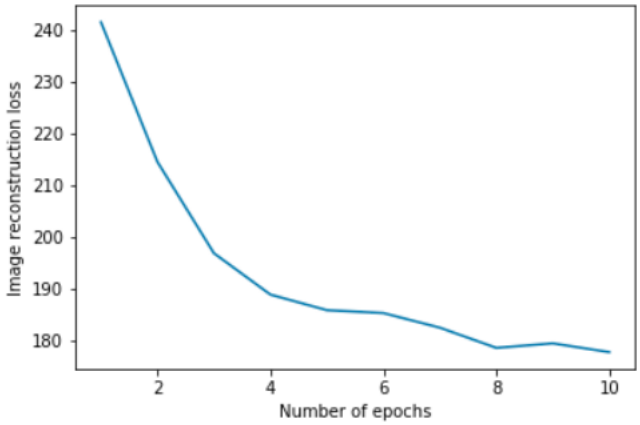


Figure 4.3: Image reconstruction loss as a function of increasing epochs

4.2.2 Results

Then, after training the VAE, t-SNE [31] is used on the embeddings of the 45 action classes training samples. T-SNE is a technique for dimensionality reduction that is particularly well suited for the visualization of high-dimensional datasets. However, figure 4.4 shows t-SNE lays a noisy 2D representation of the data. This usually means no relevant clustering has been found through the data provided and the VAE is not functional [34]. However, figure 4.5 shows the VAE succeeds in reconstructing the action sequences.

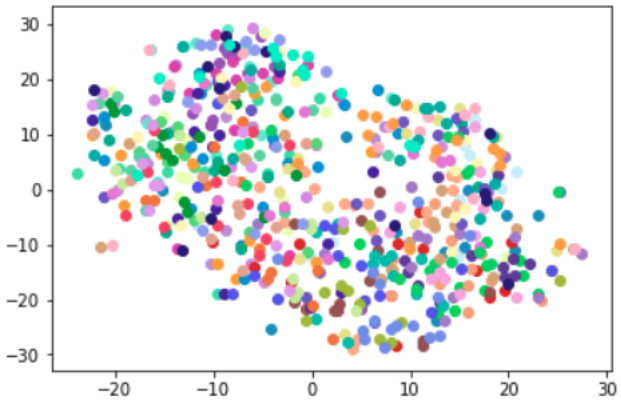


Figure 4.4: Noisy 2D t-SNE clustering on the training set embeddings with one color for each of the 45 classes

Indeed, figure 4.5 presents the original sequence input (left) and its reconstructed version (right) using the decoder. Looking at the original version, the 3 columns pattern seen is due to the input joints data being ordered as [joint 1 direction x, 1-y, 1-z, 2-x, 2-y, ...]. Hence, when the hand grasp stays fixed as in most actions, the motion in one direction involves all the joints, hence there is a high correlation between the directional values (x,y,z) of each joint which gives this copy-pasted column pattern. Similarly, whenever there is an irregular horizontal shade on groups of columns, this means the hand grasp is moving as the correlation between joints is stopped.

Looking at the reconstructed version, one can see the VAE improves its data reconstruction as training evolves and thereby, its understanding of the data it receives. At epoch 1, only a part of the data is reconstructed. At epoch 4, the VAE already makes the difference between sequence lengths by generating a shorter sample for the liquid soap action. At epoch 10, the samples are well reconstructed and the furthest away from each other in the t-SNE plot. Interestingly, the VAE starts by learning the highest numbered joints (right part of the decoded images) and later the lowest numbered joints as the left white columns only start appearing on the decoded images from epoch 4 onwards. This event makes sense with results seen earlier: the highest numbered joints correspond to the fingertips that are more discriminative about the action occurring than the joints closest to the wrist (lowest numbered joints). Hence this means the VAE prioritizes learning the most discriminative aspect about the data and later focuses on the less crucial reconstruction information.

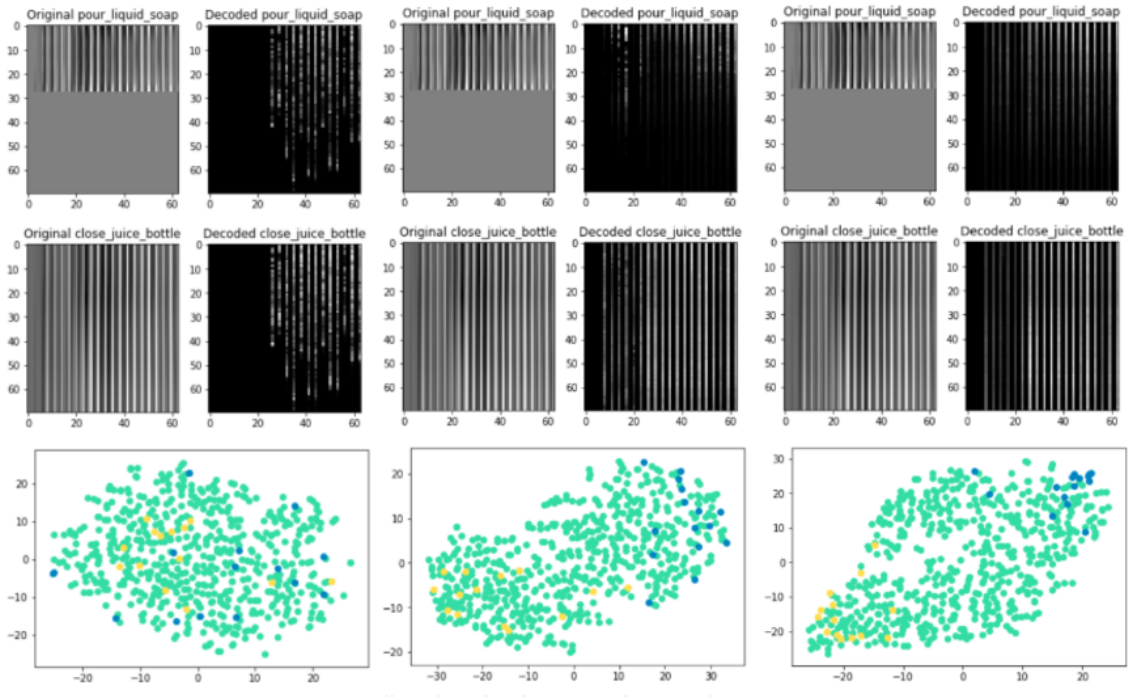


Figure 4.5: VAE decoding 2 image samples from classes *pour liquid soap* and *close juice bottle* at epochs 1, 4, 10 (left to right). t-SNE of the images Z embeddings below (yellow for top sample embedding, blue for bottom, green for all other training samples).

Before moving on to the next section and to be clear, the VAE output (reconstructed/generated data) was only used in this section to verify the VAE is functional after being trained. In the rest of this chapter's experiments, only the 'z embedding' is retrieved from the VAE in an attempt to use the latent information it encloses. Figure 4.6 clarifies what the z embedding refers to.

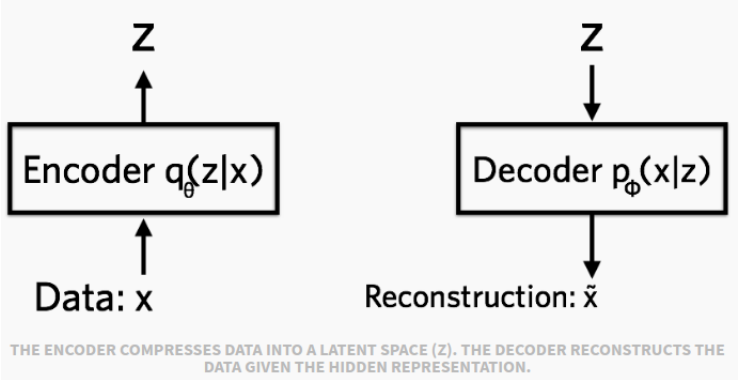


Figure 4.6: Simplified diagram of a VAE. Image borrowed from [4].

4.3 Clustering using Kmeans

Kmeans is a classic clustering algorithm based on Euclidean distance (further details in the Background section). It fits this application as it is fast, well-understood and brings a first level of understanding although there is no guarantee it is optimal to cluster VAE embeddings.

4.3.1 Parameters

Standard kmeans parameters are used in this experiment as shown in table 4.2. One of kmeans weaknesses is that the initial randomly chosen centroids can determine the final centroids significantly and lead to a sub-optimal clustering solution. As [5] states, the approximation found can sometimes be arbitrarily bad with respect to the objective function compared to the optimal clustering. Hence, the k-means++ initialization addresses this obstacle by specifying a procedure to initialize the cluster centers before proceeding with the standard k-means optimization iterations. With the k-means++ initialization, the algorithm is guaranteed to find a solution that is O(log k) competitive to the optimal k-means solution. In addition, the number of runs specifies the number of time the k-means algorithm will be run with different centroid seeds. The final results will be the best output of the consecutive runs in terms of inertia. The inertia is the sum of squared distances of samples to their closest cluster center. Then, up to 45 clusters are tried as having more clusters than action classes makes little sense. Similarly, different Z embedding dimensions are attempted.

Figure 4.7 reports the inertia for each embedding dimension as the number of clusters evolves. Since the inertia changes of an order of magnitude depending on the input dimension, the results are normalized in order to be compared. Each data series is simply divided by the largest value in the series, which is always the inertia computed with the smallest number of clusters (K=2). The results show the curve's 'elbow' is always between 5 and 10 clusters whatever the input dimension is. This means the clustering is optimal at those values as increasing the number of clusters will always reduce the inertia, until it reaches 0 when there is as much clusters as data points. Indeed, the aim is to stop increasing the number of clusters once the diminishing returns appear sub-linear (past the curve's elbow).

Table 4.2: Kmeans training parameters

Parameters	Cluster initialization	Number of runs	Number of iterations per run	Number of clusters K	Input dimension	Train:Test split
Value	$kmeans++$	10	300	2..45	$8, 16, 32, 45$	$1:1$

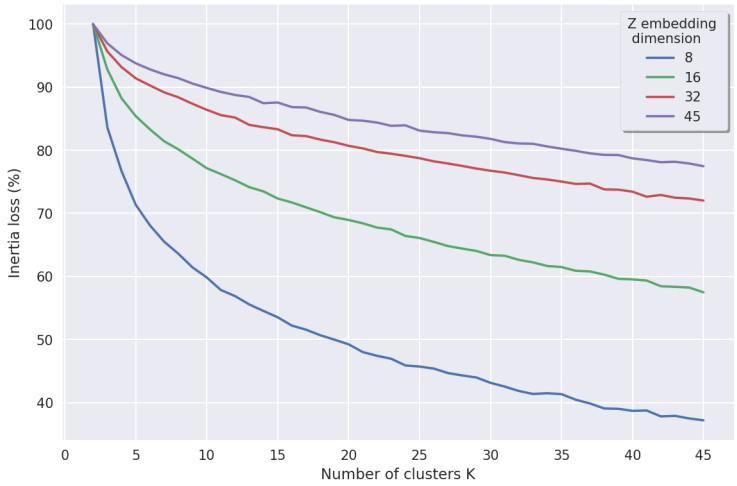


Figure 4.7: Inertia loss as a function of the number of clusters for different Z embeddings. Inertia is expressed as a percentage of the largest inertia in each data series.

4.3.2 Results

After the clusters are set using sequence embeddings from the train set, group recognition accuracy is computed with sequence embeddings from the test set, as detailed in the experiment pipeline in figure 4.1. Results in figure 4.8 show the smallest embedding dimension performs the best and closely to the other dimensions. Hence, all the following experiment results will come from clustering 8 dimensions embedding inputs.

Then, the overall performed group recognition accuracy as a function of the number of groups (clusters) is inferior to the RNN performance from manual groups. Two main reasons contribute to those results. First, kmeans performs spherical clusters because of its metric (euclidean distance) but most embeddings of actions classes probably belong to a non-linear clustering shape, hence some action classes samples end up split between multiple clusters. Second, the VAE reconstruction loss impacts the quality of results later in the pipeline.

Figure 4.8: Group recognition accuracy as a function of the number of clusters for different Z embeddings

Next, the groups created are investigated. First, figure 4.9 shows how kmeans clusters each training data point for a small number of groups (K=3). The red circles highlight which groups are the most represented in each cluster and thus, which actions define the groups the most. For example, the *write*, *flip sponge* and *open peanut butter* actions define the first cluster the most. However, watching sequences from those classes will not be enough to make sense of why those actions are grouped together. Indeed, the VAE learns a latent structure from the data without respecting a specific constraint, such as for example, modeling the data using hand grasp as a discriminative criterion. Hence, the latent structure could even be abstract to the human mind and it is normal none of the manual groups from last chapter are recognized here. It would rather be an interesting surprise if such an event happened. However, seeing how the clusters evolve as the number of groups increases could bring some insight forward.

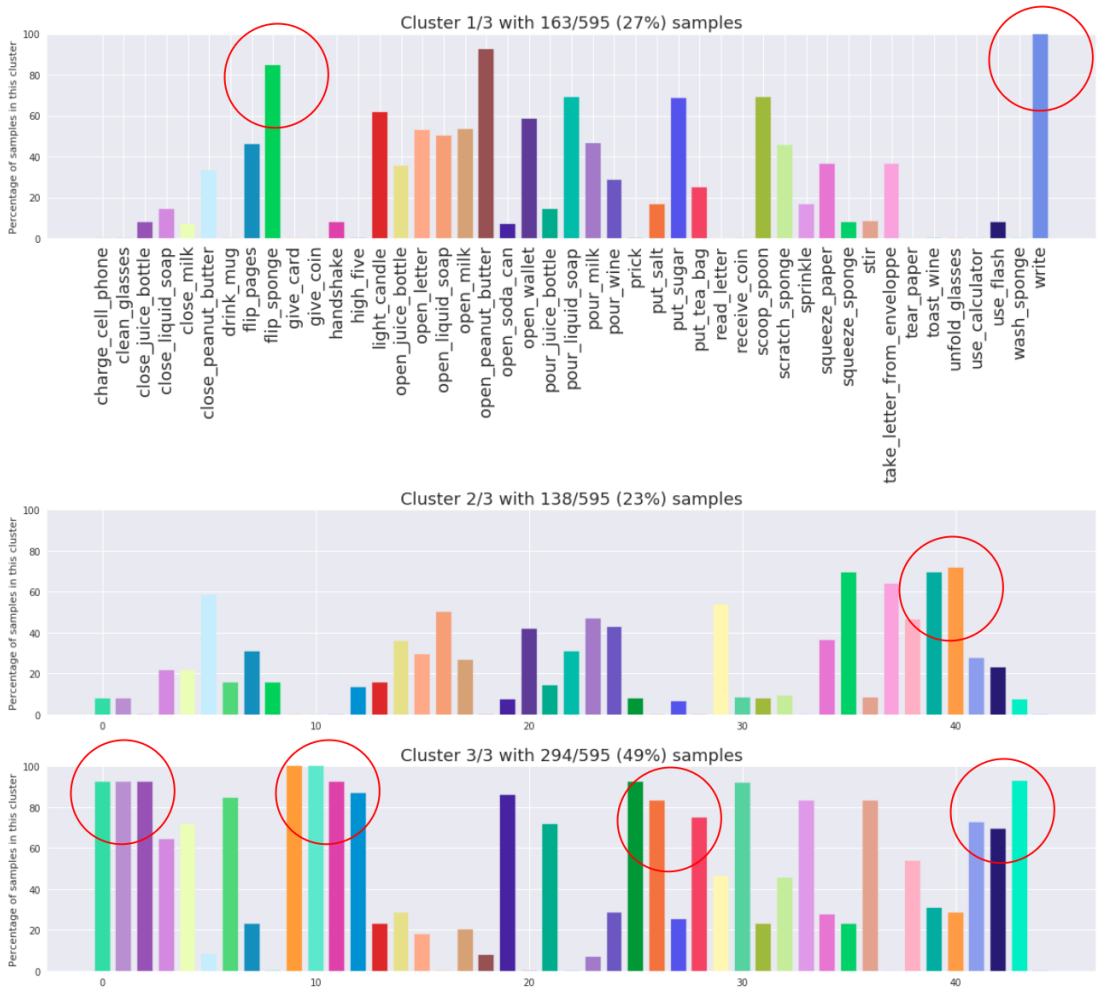


Figure 4.9: Visualization of 3 kmeans clusters. The bars represent the proportion of data points of each action class that fell in the cluster.

In figure 4.10, one can see the clusters of figure 4.9 are preserved but the second cluster now holds a defining group of actions (*put sugar* and *put teabag*) previously belonging to the third cluster of figure 4.9. First, seeing the same actions are still defining the clusters means their embeddings have an information in common that the VAE uses to reconstruct the sequences. The fact they are clustered together is not random or some noise clustering. Second, this highlight some hierarchical classification where, inside each cluster, there can be some further differentiations between the actions thanks to the VAE embedding. However, some other actions constantly stay together such as *give coin* and *handshake*, which were in the same manual group as both action involves a hand with a similar grasp and motion going towards another hand.

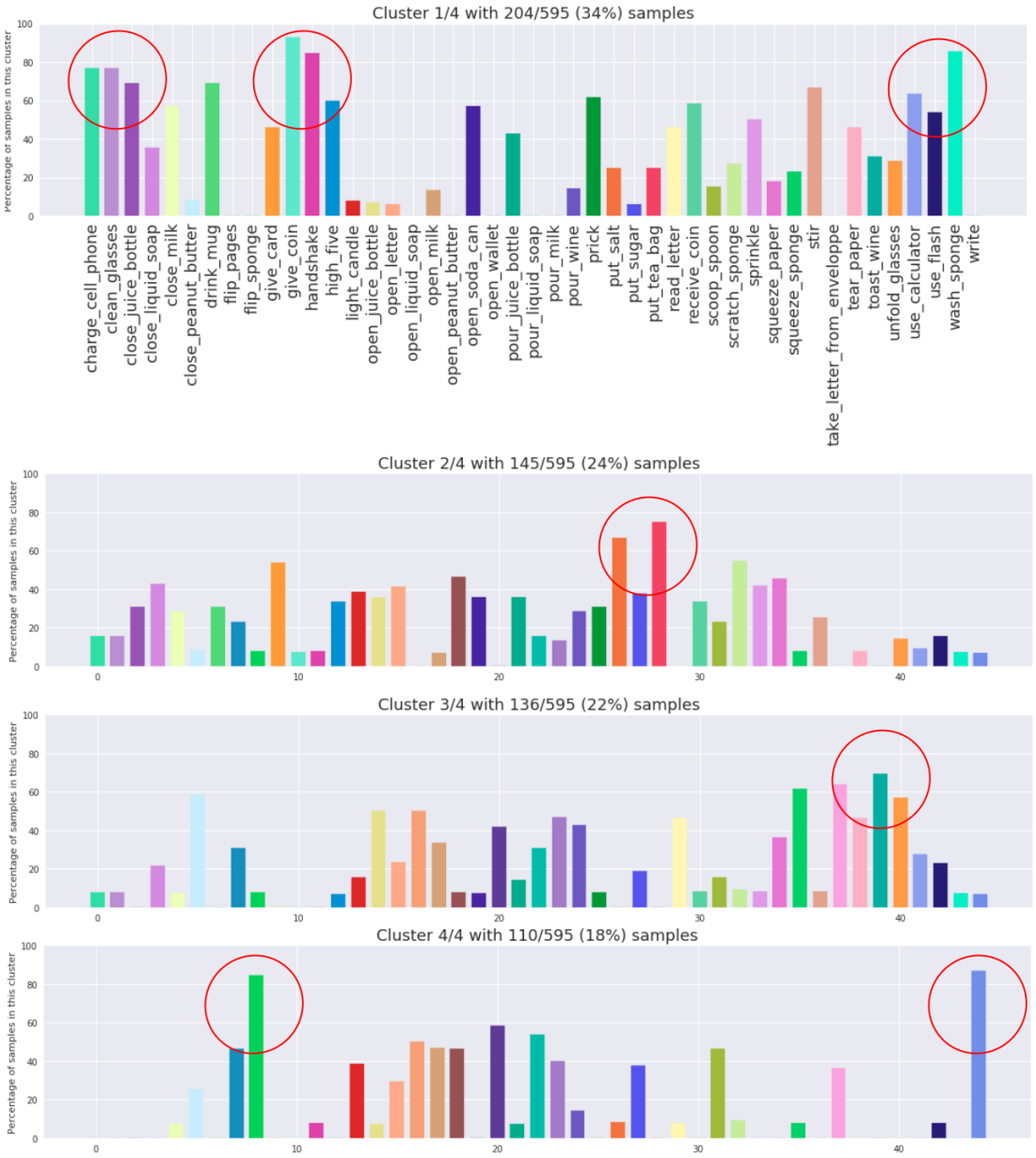


Figure 4.10: Visualization of 4 kmeans clusters.

Increasing the number of clusters further in figure 4.11 shows similar results with still the same actions leading each group, except a few actions previously split between clusters have now got a cluster of their own where they are a defining action. For example, *flip sponge* has stopped teaming up with *write* and has now a cluster of its own with *take letter from enveloppe* which was previously split between two clusters. Hence, now that an additional cluster can be built, the algorithm finds it optimal to group those two actions together instead of previously splitting them between clusters. This might mean the two actions are coded in the VAE with a discriminative information that is the fifth most used to reconstruct any action.

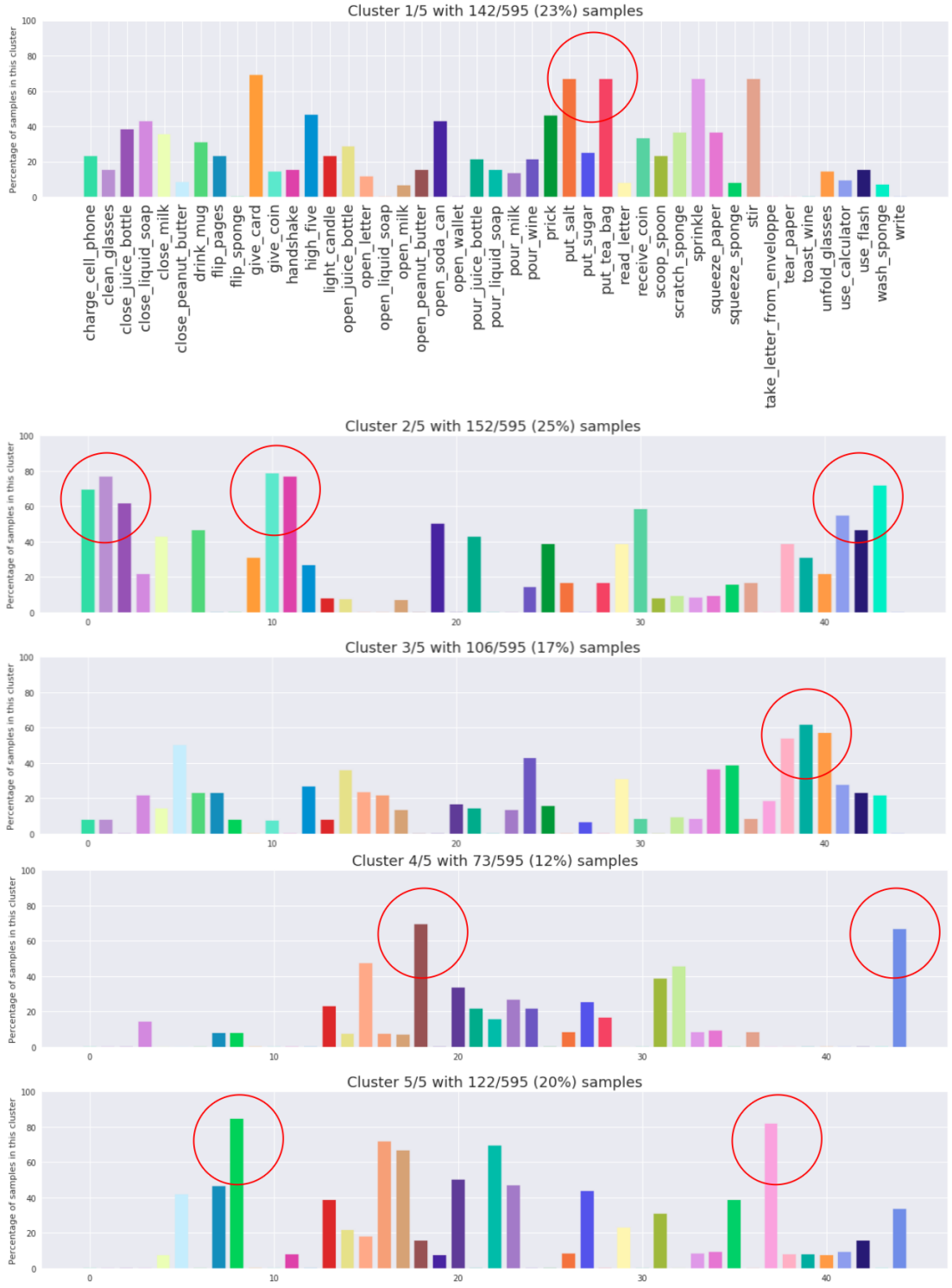


Figure 4.11: Visualization of 5 kmeans clusters.

To sum up the observations across the three figures, a few number of actions constantly define the clusters. This indicates those actions have a certain shared spherical coherency across their different z embeddings which allows to build a strong cluster around them. Indeed, the VAE has trained itself in a way that makes those actions have closely coded embeddings, which might imply those actions share some kind of latent information the VAE uses.

Figure 4.12 shows more clearly which clusters each action belongs to, along with the proportion of data points from this action present in the cluster. This allows to see in increasing order which

are the defining actions of the clusters.

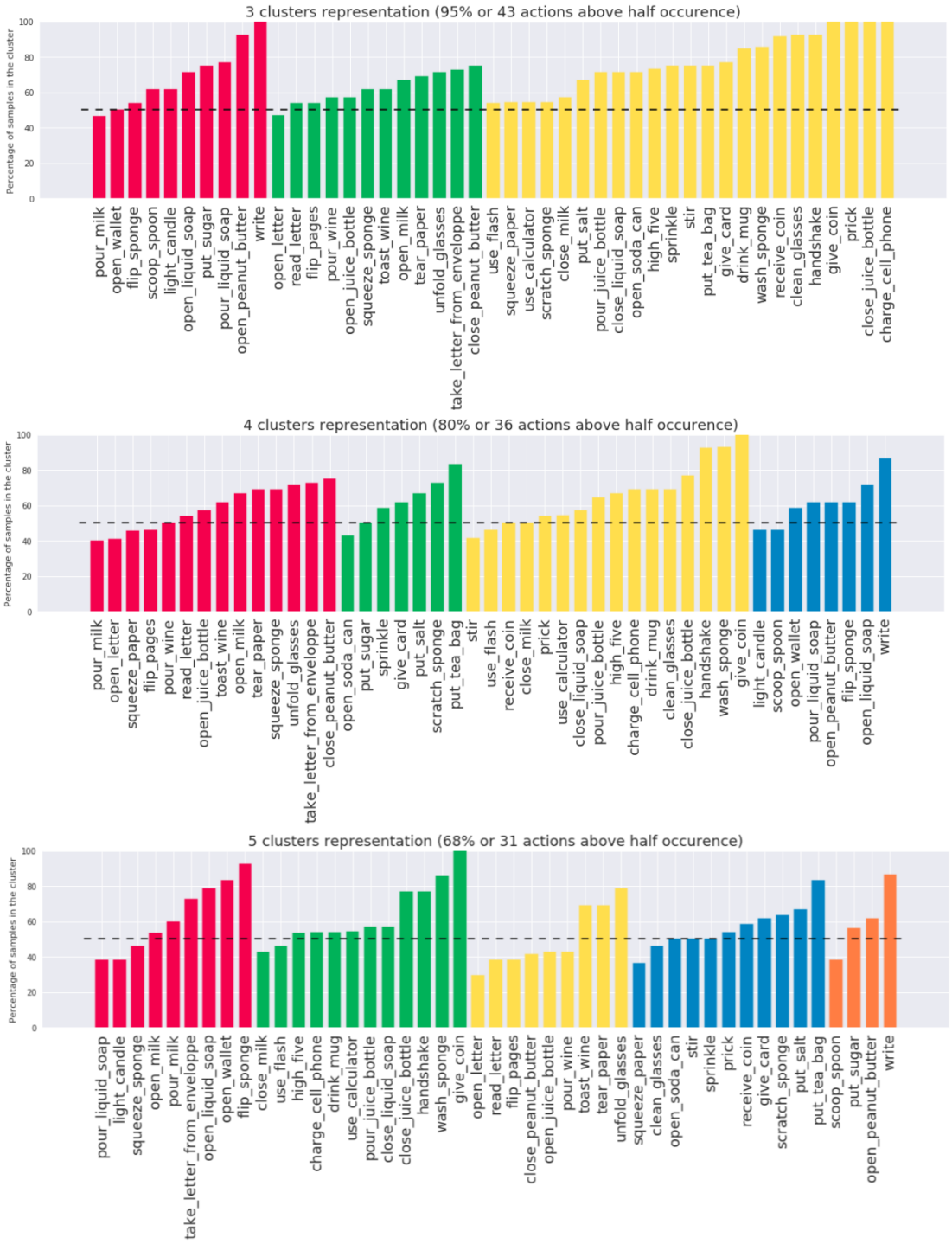


Figure 4.12: Visualization of the proportion of presence of each action in the cluster they belong to for 3,4 and 5 clusters. Each color represents a cluster and an action belongs to the cluster where a majority of its data points are.

Lastly, increasing the number of clusters to 8 shows many actions stay the cluster lead such as *flip sponge* or *give coin*. However, some less discriminative clusters start being created such as the one where *stir* is with an occurrence of its data points below 50%. Thus, those new clusters that only enclose a small part of the data points of even the leading actions can be considered as

noisy (useless) clusters since they regroup actions through a latent information that is not present in every one of their samples. Therefore, this latent information is better left unused as it is not constantly present when the actions is executed and thus can not be used as a means to recognize the action. For example, this useless latent information could be the angle in space or the speed the action is executed at, which only concerns a few data points of every action.

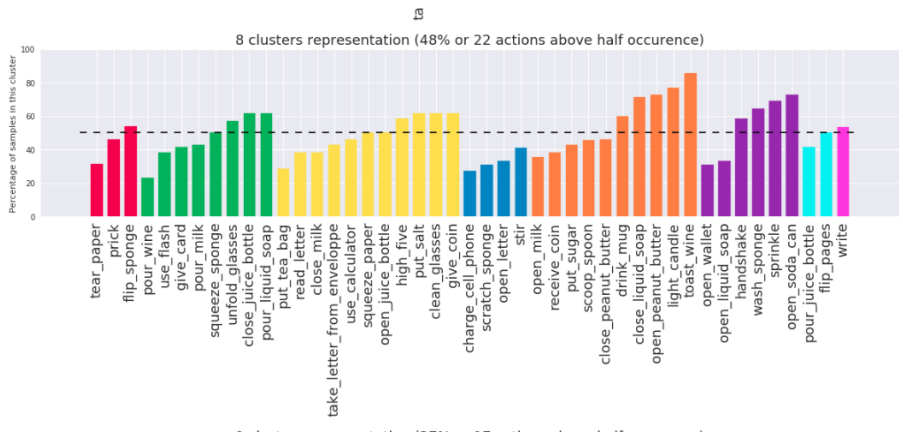


Figure 4.13: Visualization of the proportion of presence of each action in the cluster they belong to with 8 clusters

4.4 Conclusion

To conclude, one can definitely appreciate the VAE extracts relevant information from the action sequences as clustering those embeddings shows some groups of actions are constantly clustered together. However, kmeans is limited in the insight it can bring to understand what information is inside those embeddings. Concerning the method, it indeed builds groups automatically but the group recognition accuracy is worse than if those groups were handcrafted.

Further steps could include:

- Incorporating the VAE and Kmeans in a feedback loop where poor group recognition accuracy is accounted for in the VAE loss function. Hence, the VAE embeds the sequences with latent information that can be clustered easily and brings to a better group recognition accuracy. In the end, this method tries to link actions through abstract latent information that a human might not understand. Hence, it is more important to constrain the system to learn this latent information than to work on making this information understandable.

- Embedding each hand pose frame instead of each sequence of hand pose. This is similar to embedding words instead of sentences in natural language processing and could allow the VAE to reduce its reconstruction loss by coding the hand pose sequences more precisely. This would also allow to evaluate how much of each action's hand pose is present in another action.

- Investigating the discriminative power of each direction(x,y,z) by embedding only one direction and evaluating how the clustering changes.

- Using a Variational Recurrent Autoencoder [13]. It can be used for large scale unsupervised learning on time series data, mapping the time series data to a latent vector representation.

Chapter 5

Conclusion

To conclude, the best individual and group recognition accuracies in this project are achieved through the handcrafted groups. Those groups also allow to share insight on the hand pose data static and dynamic discriminative patterns. Finally, using unsupervised learning methods on the dataset does reveal the existence of more of those features but still makes difficult their extraction and intuitive understanding.

Appendix A

Chapter 3 additional figures

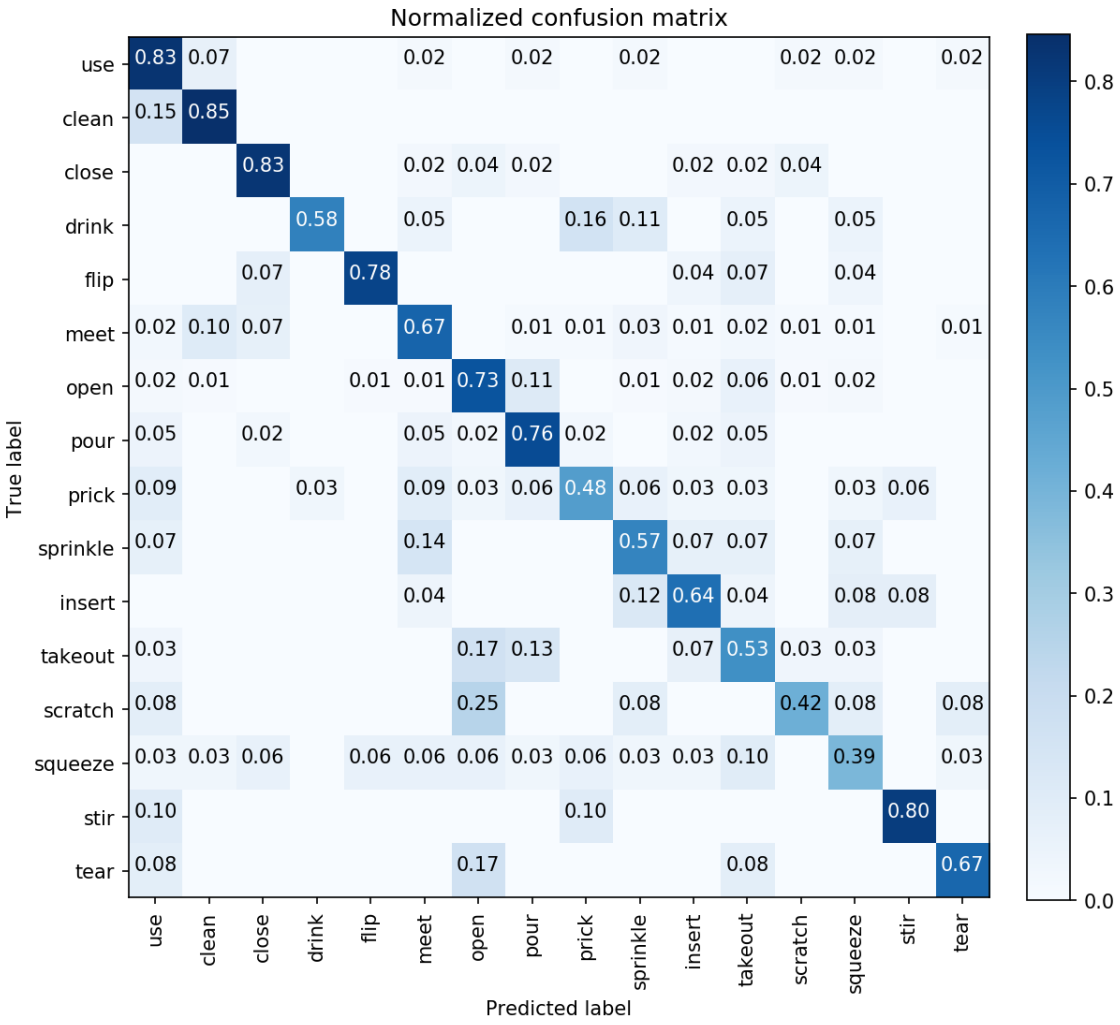


Figure A.1: 12 Object grouping confusion matrix on the trained RNN

Appendix B

Github repository

The code used to build the experiments in this report is available at: https://github.com/natoucs/FYP

Bibliography

[1] Understanding lstm networks. 2015.

[2] Kullback-leibler divergence explained. 2017.

[3] Wikipedia: Multi-task learning. 2017.

[4] J. ALTOSAAR. Tutorial - what is a variational autoencoder ? 2017.

[5] D. Arthur and S. Vassilvitskii. K-means++: the advantages of careful seeding. In *In Proceedings of the 18th Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007.

[6] R. Balestriero. Neural Decision Trees. 2017.

[7] H. J. Chang and G. Garcia-Hernando. Spatio-temporal hough forest for efficient detection–localisation–recognition of fingerwriting in egocentric camera. 2017.

[8] J. Chung, S. Ahn, and Y. Bengio. Hierarchical multiscale recurrent neural networks. 2016.

[9] B. Dai, Y. Wang, J. Aston, G. Hua, and D. P. Wipf. Veiled attributes of the variational autoencoder. 2017.

[10] Digi-capital. Augmented/virtual reality revenue forecast revised to hit 120 billion dollars by 2020. 2017.

[11] C. Doersch. Tutorial on Variational Autoencoders. June 2016.

[12] Y. Duan. One-shot imitation learning. 2017.

[13] O. Fabius and J. R. van Amersfoort. Variational Recurrent Auto-Encoders. 2014.

[14] S. R. Fanello. One-shot learning for real-time action recognition. 2015.

[15] K. Frans. Variational autoencoders explained. 2016.

[16] G. Garcia-Hernando. Transition hough forest for trajectory-based action recognition. 2016.

[17] G. Garcia-Hernando, S. Yuan, S. Baek, and T. Kim. First-person hand action benchmark with RGB-D videos and 3d hand pose annotations. 2017.

[18] T.-K. K. Guillermo Garcia-Hernando. Transitions forest: Learning discriminative temporal transitions for action recognition and detection. 2016.

[19] A. Jahn and K. David. Improved activity recognition by using grouped activities. 2016.

[20] A. Karpathy. The unreasonable effectiveness of recurrent neural networks. 2015.

[21] D. P. Kingma and M. Welling. Auto-encoding variational bayes. 2013.

[22] B. Lakshmanan and R. Balaraman. Transfer learning across heterogeneous robots with action sequence mapping. 2010.

[23] Y. LeCun and C. Cortes. MNIST handwritten digit database. 2010.

[24] A.-A. Liu. Hierarchical clustering multi-task learning for joint human action grouping and recognition. 2017.

[25] A. Lopes. Transfer learning for human action recognition. 2011.

[26] F. Mohr. Teaching a variational autoencoder to draw mnist characters. 2017.

[27] B. Ni. Motion part regularization: Improving action recognition via trajectory group selection. 2015.

[28] D. Roy, P. Panda, and K. Roy. Tree-cnn: A deep convolutional neural network for lifelong learning. 2018.

[29] Z. S. Seungryul Baek. Kinematic-layout-aware random forests for depth-based action recognition. 2017.

[30] J. G. Umar Iqbal, Martin Garbade. Pose for action – action for pose. 2016.

[31] L. van der Maaten and G. Hinton. Visualizing data using t-SNE. *Journal of Machine Learning Research*, 2008.

[32] J. Walker, K. Marino, A. Gupta, and M. Hebert. The pose knows: Video forecasting by generating pose futures. 2017.

[33] Z. Wang, J. Merel, S. E. Reed, G. Wayne, N. de Freitas, and N. Heess. Robust imitation of diverse behaviors. 2017.

[34] M. Wattenberg, F. Viégas, and I. Johnson. How to use t-sne effectively. *Distill*, 2016.

[35] X. X. Wentao Zhu. Co-occurrence feature learning for skeleton based action recognition using regularized deep LSTM networks. 2016.

[36] M. Wray, D. Moltisanti, and D. Damen. Towards an unequivocal representation of actions. 2018.

[37] H. J. C. Youngkyoon Jang, Seung-Tak Noh and T.-K. Kim. 3d finger cape: Clicking action and position estimation under self-occlusions in egocentric viewpoint. 2015.

[38] T.-K. K. Zhiyuan Shi. Learning and refining of privileged information-based rnns for action recognition from depth sequences. 2017.