Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Learning corrective models for multi-step action from videos

Swaroop Bhandary K

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul G. Plöger
M.Sc. Alex Mitrevski

January 2019

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Swaroop Bhandary K

Abstract

Teaching robots to perform tasks is complicated as there can be multiple ways to perform the tasks and also due to the variability in the environment. Learning from demonstration is one of the most common methods to counter most of these issues; however, learning from demonstration itself has several challenges. Two of the significant ones are segmentation of the entire task into modular subtasks and generalization over multiple demonstrations. With this project, we have tackled these two challenges by using state of art deep learning methods to perform segmentation and using a temporal constraint learning approach for generalization.

For segmentation, 3D-CNNs have been used to recognize the subtasks/actions performed in the demonstration. For generalization, a temporal constraint learning approach has been used where initially the possible action sequences are very constrained; these are then relaxed given more demonstrations.

The 3D-CNN model was evaluated on the UCF dataset and the MP-II cooking activities dataset. A test accuracy of *48.19*% on UCF-101 dataset was achieved; however, the 3D-CNN model was not able to learn the cooking action present in the MP-II cooking activities dataset. The inability to learn the cooking action can be attributed to the scarcity of bench action datasets in the domain of cooking. An improvement of 10% in test accuracy was noticed by fine tuning the weights learned for the UCF-101 dataset to the MP-II cooking action dataset. The constraint learning approach developed for learning the temporal constraints was evaluated on various use cases such as coffee making, mashed potato preparation, and sandwich preparation and it was able to generalize well for all the above mentioned use cases.

Acknowledgements

I would like to thank my supervisors, Prof. Dr. Paul G. Plöger and Alex Mitrevski, for their support and guidance throughout the project. I would like to thank Alex Mitrevski for his continuous support and feedback without which it wouldn't have been possible to complete this R&D.

Contents

1

Introduction

Any task that a human performs in his day to day life contains numerous actions. Take for example a simple task of boiling an egg, which involves holding an egg, turning on the stove, boiling water, placing the egg in boiling water, waiting for some time and turning off the stove. These subtasks seem trivial to us as we would have encountered some of them before and we would have the knowledge of the steps involved in performing them. But for robots, even such simple task seem very complicated; this is because they have to learn to perform this task by understanding every tiny detail.

Programming a robot[1] to perform such multi step tasks is very complicated due to the variability in the tasks (various ways to perform the same task) as well as the environment (in this example, the kitchen setting). Hence learning seems to be a promising approach as the robot can cope with varying environment setting and they can improve their performance over time.

Various approaches involving learning have been used to teach robots to perform tasks. Most common approaches are learning from demonstration(lfd) [4] [10] [6] [39] [24] and reinforcement learning [37]

We focus on learning from demonstration approaches as reinforcement learning has slow convergence rate and it is difficult to decide the reward functions for input modalities such as video streams.

[1] robot and agent have been used interchangeably throughout this report

In learning from demonstrations, the process of learning a complex task is usually broken down into two stages. In the first stage, the agent is provided with demonstrations of the actions/subtasks involved and the agent learns to perform these sub-tasks. In the second stage, demonstrations of the entire tasks are provided so that the agent can learn the proper sequencing of the actions/sub-tasks. The assumption in this approach is that the set of actions/sub-tasks required to complete the given task is known.

We follow this approach and assume that the robot is capable of performing the sub-tasks involved and focus more on learning the proper sequencing of these actions/sub-tasks from the video demonstrations. In other words, we try to learn the temporal sequencing of actions to complete the given task. By learning the correct temporal sequencing we aim to build a validation model using which a robot should be able to perform validation and be able to provide suggestions about corrective actions in case of any misses.

We focus on learning from videos rather than from human demonstrations for the following reasons:

- Videos are a simpler and more versatile source of demonstrations.
- Learning task from videos will lead to models that are more generalized. The reason being different people perform the task in varied environment setting.
- A demonstrator might behave out of habit. For example, when preparing coffee, if the demonstrator always puts sugar into the cup before coffee powder, the robot might learn an artificial constraint that sugar has to be put before coffee powder which is actually not the case. When learning from videos, different people perform the tasks hence the possibility of such habitual behaviors being repeated is very less. Hence providing novel demonstrations for the robot to learn from.

1.1 Use case

To understand the problem statement and coming chapters we briefly explain the preparing coffee use case here. In-depth explanation of all the use cases used to validate the working of our model have been explained in chapter 4.

1.1.1 Preparing coffee

Below are the basic steps involved in making a cup of coffee:

1. Turn on the stove.
2. Keep vessel on the stove.
3. Pour water into the vessel.
4. Boil.
5. Put sugar into the cup.
6. Put coffee powder into the cup.
7. Pour hot water into the cup.
8. Stir.

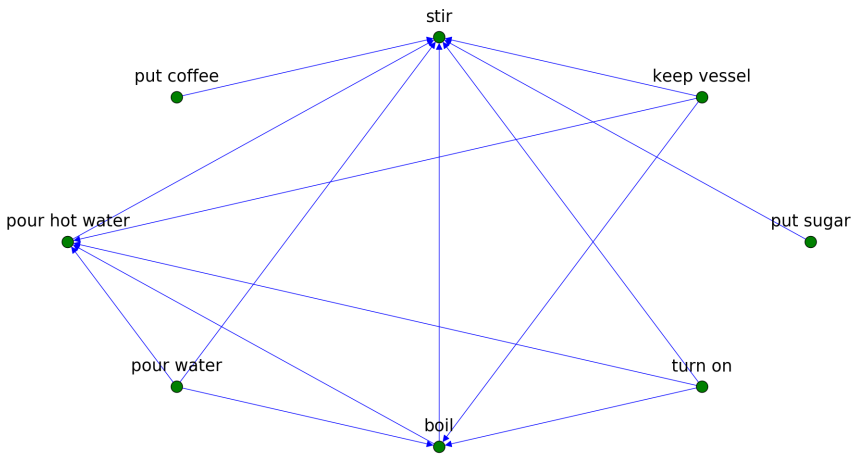

Figure 1.1: Constraints for coffee making.
Arrow from $action_i$ to $action_j$ indicates that $action_i$ has to be performed before $action_j$

We can see that there are multiple ways of preparing a cup of coffee. Step 1-3 can be performed in any order. Similarly, step 5-7 can be performed in any order. But step 4 has to be performed after step 1,2 and 3. So our model should be able to find all these constraints and be able to perform validation. We aim to learn the constraint graph as shown in fig 1.1.

1.2 Problem Statement

There are various videos on the world wide web with detailed steps on performing a specific task. Take for example cooking; there are various cooking video tutorials in YouTube that one can use to learn to prepare a specific dish. These videos are large sources of information. But when it comes to the field of machine learning, there are currently few weak models that can process entire videos and extract features from them that can be used to build corrective models. As a result, it is difficult to leverage these sources of information for the purpose of supervising/teaching. The proposed work addresses this issue. In this work, we are particularly interested in learning how to extract the relevant information from videos and in building a high level representation of the task which the robot can use to perform validation.

The process of learning from demonstration consists of the following steps: providing demonstrations, segmentation of the complete task into modular subtasks/actions, state generation and finally generalization over various demonstrations.

With this work, we aim to tackle the problems related to segmentation of tasks into modular subtasks/actions and task generalization. By segmentation, we plan to determine the action that is being performed and extract the action sequences performed in the demonstration. Deep learning models which are state of the art in action recognition have been used to recognize the actions performed in the demonstration. The input to the generalization module is multiple action sequences from various demonstrations. With the generalization module we plan to build a high level representation of the task which takes into consideration all the demonstrations provided to it.

So with this project we aim to meet the following objectives:

- Learn nominal execution models of sequential activities from videos.
- Extend the validation capabilities so that it can also suggest corrective actions.

1.3 Relevance

The relevance of this work is as described below:

- Ease of teaching task to the robots: A human need not demonstrate the task to the robot rather the robot itself can watch videos from web world web and learn to perform these tasks.

- Easily transferable to other robots: The model built is a high level representation of the given task. The assumption is that the robot knows to perform primitive actions. Hence it can be easily transfered to other robots.

1.4 Challenges and Difficulties

- Extracting relevant information from videos such as the action being performed: Action recognition by itself is a complex task. The main reasons are

  - Dealing with temporal domain: Each action spans a certain number of frames and all these frames carry valuable information. Hence, the models for action recognition need to deal with the extra temporal domain as it is usually not possible to determine the action given a single frame.

  - Computational complexity: As mentioned above, the models for action recognition have to deal with the temporal dimension. Hence, this has to be encoded into the network architecture which leads to increased computational complexity.

  - Scarcity of benchmark video datasets: The benchmark datasets for video recognition are HMDB51 [22] and UCF101 datasets [8]. The number of sample clips per video on an average is around 100 which is much lesser when compared to the image dataset such as Imagenet which has around 1000 samples per class. Also, there is no benchmark dataset for cooking scenario. The datasets available are highly biased towards certain actions.

- The high level task representation used should be capable of performing validation on the given task and hence should be able to encode various ways of performing the tasks.

- Dealing with the temporal domain: Each action takes different number of time steps to complete. Hence, this has to be taken into consideration when building the task representation.

1.5 Assumptions

- All the actions needed to complete the task are known and can be recognized by the action recognition model. The action models which are of interest have been described in section 3.2.1.
- The preconditions and post conditions for each action is known.
- Actions can be recognized given only 16 frames.
- When learning the high level representation of the task, we assume that only the correct way of performing the task is shown.
- Only proper sequencing of actions in the form of temporal constraints is learned from the videos.

2

State of the Art

The easiest way for an agent to learn a given task is to perform it in the exact same manner as done in the demonstration. This is done in behavior cloning[33]. In this approach, the agent receives demonstrations that consists of a sequence of observation-action tuples and it tries to learn a function that maps observation to actions. Learning of the function is done by using either a regressor or a classifier. This method is powerful as it can imitate the expert demonstrator without having to interact with the environment. The disadvantage of this method is that the approach fails when the agent comes across a state that it has never seen during the expert demonstration as the agent does not know what action to perform in this state and it is difficult to recover once it is lost. Behavioral cloning should be used when the deviation from the expert trajectory is not very costly and when the expert demonstration cover that state space pretty well [39].

Reinforcement learning is another approach that is used to teach tasks to robots. The main disadvantage with reinforcement learning is it's slow convergence rate and difficulty in determining a reward function for input modalities such as video. The problem of slow convergence rate has been tackled by first using learning from demonstration to determine the basic policy from the expert demonstration and once the policy is learned, inverse reinforcement learning (IRL) is used to approximate the reward function. Once this is done, the agent uses the learned reward function to better the initial policy [6]. While IRL can in theory be applied to any problem, it is proven to be difficult to apply IRL when the input is high dimensional [24]

To overcome this issue, [24] has come up with imitation from demonstration. Here, explicit state-action pairs are not provided but multiple time synchronized video demonstrations of the same task on different contexts are provided. Here context can mean changes in view point, background, positions etc. Let the given demonstrations be

$$D_1, D_2, ..., D_n = [O_0^1, O_1^1, ..., O_T^1], [O_0^2, O_1^2, ..., O_T^2], ..., [O_0^n, O_1^n, ..., O_T^n]$$

where O_i^j is the ith observation/frame in the jth demonstration.

The authors train a context translation model to translate context from D_i to D_j given O_0^j. The model is trained in a supervised manner with squared error between the encoded version of translated observation and the encoded version of the actual target context observation. The encoded version is obtained by using the representation in the final layer of a CNN. Deep reinforcement learning is then used to learn the actions in the translated encoded observations. Hence, in this method we don't need the action sequences explicitly. But all the demonstration must be time synchronized which is a drawback in this approach.

In [44], the authors have used a grammar tree structure similar to ones in linguistic domain to store the action sequences. The authors have used two CNNs to extract the required information from videos. One CNN is used to decipher the action that is being performed and the other is used to determine the grasp type. Six grasp types and ten actions have been considered. To avoid the complexity of dealing with temporal domain, the authors have used a probabilistic approach to determine the action. The probability of action given the objects being manipulated is calculated $P(Action|Object_1, Object_2)$ and the action which is most likely is then picked up. The data extracted for each frame will be a tuple containing the following values: $LeftHand : GraspType_1, Object_1; RightHand : GraspType_2, Object_2; Action$. Once the required information is extracted from the video, a viterbi probabilistic parser is used to represent the task as a hierarchical recursive tree structure. By reverse parsing this tree, the robot is able to perform the learned task. However, this tree structure while good enough to perform the learned task is not strong enough to perform validation as this structure is not capable of encapsulating knowledge from multiple demonstration. i.e. there is no approach to combine the output trees

of two demonstrations.

In our work, since we want to perform validation on given task, we want to determine if a given sequence of actions is valid or not. This could be done by learning the preconditions for each of the primitive action from the video, learning the constraint on the sequencing of primitive tasks or by building a probabilistic model to determine the action transition given a particular state. The first two approaches use a deterministic model (similar to models in the planning domain) whereas the latter uses a non deterministic model (Markovian models). We concentrate more on the deterministic models where we try to determine the allowed execution sequence of primitive action to complete the given task.

2.1 Deterministic Models

2.1.1 Precondition learning

In the papers mentioned below, the authors have defined the possible action set but the precondition and effect of each of these actions is learned from the demonstration.

In [4], the authors believe that the robots should be capable of reasoning on a symbolic level and appropriately chain the primitive actions to perform new tasks. This has been done by providing the robots the ability to determine the preconditions and post condition for each action directly from the user demonstration. An initial set of features are chosen. The feature set could be properties such as position, orientation etc. The model then tries to decide which features is a precondition by checking the feature value variance at the beginning of the experiment across various demonstrations. If the variance exceeds a predefined threshold, it is determined as irrelevant. For discrete case, binary cross entropy has been used to determine the variance whereas for the continuous case, k-means clustering is first performed and then a distance metric (either euclidean distance or angular difference between rotations) is used to determine the variance. In the same manner, the effect of each action is also determined. A symbolic planner can then be used to plan a complex manipulation task.

In [5], the authors have used imitation learning to learn to perform primitive tasks.

Sensometric data is used to learn how to perform these primitive actions. Once the agent learns to performs these primitive tasks, visuospatial learning (VSL) is used to then learn the precondition and effect of each of these actions. The input for the VSL module is the camera feed. The VSL captures two main observations: pre-action observation and post-action observation which are used to determine the precondition and effect. VSL then creates a symbolic representation and also extracts a feature vector (SIFT) for the objects detected in the observation frame. It then extracts the pose of these objects. These are then used to determine the preconditions and post conditions of the action. Hence by using this approach, the agent is able to learn how and when to perform the primitive actions. Now, a symbolic planner can be used to determine the action sequence to reach a specific goal state.

The preconditions and post conditions for actions have been determined by picking up features such as positions, orientation of objects with low variance between the values in various demonstrations. Hence the implicit assumption in the papers explained above is that the environment where the task is being performed is the same. Also, these methods require pose data which is also used for both precondition and post condition list determination. Hence these approach do not fit our requirements as we are trying to learn from video and cannot guarantee the environment where the demonstration is being provided is the same. Hence it is not possible to use these approaches in our work.

In [25], the authors have assumed that the agent initially knows only a weak domain model. Hence the robot is capable of identifying objects; is aware of the relevant object properties and relations; and also knows what actions can be performed but not the entire action model i.e. it does not know the preconditions and effects of the action. The authors have described a method to extract all these data from noisy and incomplete observations. STRIPS operators cannot deal with noisy data. Hence the authors have developed a two step approach. Firstly, the authors try to model the actions using disjunctive normal form (DNF) kernel voted perceptron classifiers which are more robust to noise and can work with incomplete data. The training data for the classifier is a sequence of state and actions where each state is a set of fluent expression. Here fluent expression refers to a parameterized predicate. All the training examples are converted to a fixed sized vector with each bit representing if a fluent expression is true or false. Once this is done, a change vector is calculated

which is the difference between the fixed sized vector of the current state and the fixed vector of the state after performing the action. The classifier is then trained to classify the value of a single bit of difference vector by providing the corresponding precondition vector as input. Now by using the support vectors that have been learned in the previous step the authors have determined if the fluent corresponding to that bit will change if the action is performed. The algorithm to extract pre and post condition rules from the learned support vectors has also been described in this paper.

In [43], the authors have tried to learn action model given a set of successfully observed plans. Also, the authors have not assumed complete state observability. So each plan describes an action sequence and partial intermediate state information. This approach can also be used when no intermediate state information is provided. This method, however, requires that all execution sequences are correct. Given the action sequence with the parameter list, three different constraints (action, plan, information) are determined for the task from the action sequence. Action constraint holds that any precondition for an action cannot be present in the add list of that action and also that the delete list of an action is a subset of the precondition list. In plan constraints, constraints based on temporal ordering of actions are present i.e. it explains why a specific action is performed before or after a specific action. It might be because a specific action a_2 contains in its precondition list a precondition that is present in the add list of a_1 and no other action between a_1 and a_2 has this in the add list. After this, a frequent-set mining algorithm has been applied to determine the weights for each of these constraints. It is then fed to MAX-SAT solver. A set of constraint that have been satisfied in MAX-SAT is considered as the precondition or post condition of a particular action.

In [46], the authors have relaxed the assumption of having only correct execution sequences and described an approach to deal with noisy action sequences as well. The approach has been named Action Model Acquisition from Noisy plan traces (AMAN). The authors try to find the model which best describes the observed plan traces. The authors first determine all the possible predicates and action schemas from the given plan traces. Once this is done, all the models are enumerated to generate a candidate model list. The authors have then described the relation between the state, current action, observed action and domain model using a graphical model.

This graphical model is then used to predict the actual execution sequence and domain model given the noisy action sequence. Hence the goal is to predict $P(t, m|t^0)$ where t is the actual plan trace, t^0 is the observed plan trace and m is the domain model. Each candidate model is parameterized by weight and posterior distribution of an action sequence (described in a log likelihood manner) in which each action is described by a set of features. The learning of the weights and features is done using a policy gradient method which tries to maximize the reward which is the product of percentage of actions performed successfully in the plan trace and percentage of prepositions that have been satisfied in the goal state. During each iteration, a model is picked based on the weights and then an action trace is sampled using the model, state and given action. This trace is evaluated using the reward function and then the parameters are updated depending on the reward. The model with the highest weight at the end of fixed number of iterations is the required model.

The approaches [25], [43] and [46] require as input the parameterized predicate/action sequences. Given these action sequences, they try to predict the action models which is most likely to have produced the given action sequences. The above mentioned approaches can be used in our case as well. However, these approaches cannot be used in a online manner hence it is not possible it is difficult to extend the learned model for more demonstrations.

2.1.2 Determining action sequences for known primitive actions

The papers discussed below fall into the second category i.e. determining the proper sequence of performing the primitive actions. This has been done by either learning a temporal constraint on the sequencing of actions or by building a higher level task representation where the edges define the sequencing constraints. Hence all these follow the common assumption that an action model with precondition and effect is already present.

In [10], the authors have performed constraint identification by performing multiple demonstration. For some tasks, the order of execution of steps matter and for some it does not. These temporal dependencies cannot be captured when the agent is provided with only one demonstration. So when the agent is provided with

a single demonstration, the agent always performs the action sequence in the exact same manner. When more demonstrations are provided, it might learn that few constraints are contradicting in which case these constraints are removed. Hence, given multiple demonstrations of a task, the agent will be able to perform the given task by creating an action sequence of its own with the knowledge it has acquired from the multiple demonstrations provided to it. Hence, it can find novel ways of performing a task as long as none of the constraints are violated.

In [27], the authors have discussed an approach where the robot learns the proper ordering of primitive behaviors which the robot already knows to perform. Hence, a high level representation of the given task is learned on top of the capabilities that the robot already has. A representation similar to a behavior based network has been used. Each node represents a behavior that the robot can perform and the links between each node represents precondition-postcondition dependencies. The activation of each behavior is dependent on the environment conditions as well as the post conditions of the previous behavior. Three types of precondition edges have been defined: permanent precondition, enabling precondition and sequential precondition. Given the demonstration, the agent tries to learn the various kinds of precondition edges between various behaviors and also the sequence in which the behaviors are executed.

[26] is an extension of [27]. In this paper, each action node is split into two nodes: abstract node and primitive node. Primitive nodes define how primitive actions are performed whereas the abstract nodes define the corresponding precondition and effect for that primitive action. It can be seen that abstract nodes are basically environment states which are represented in a predicate-like form. This has done to provide more generality to primitive actions as the preconditions might vary for different task. This has been described in [28]. The agent observes the human during the demonstration and when the environment state matches the effect of given action described in the abstract node, it knows that, the given action is performed. Depending on when each of the behavior fires different temporal dependencies between them is found. The robot is also made to perform the task during the demonstration and depending on the performance of the robot, the demonstrator can either provide feedback or perform another demonstration. Task generalization has also been discussed in this paper. A dynamic programming approach of Least

Common Subsequence has been used to determine the common steps in the various demonstration and this is then used to create a more generalized topology.

In [13], the authors have addressed the problem of learning a constraint network of a given task. However, this requires a large number of demonstration. The authors have addressed this issue by using active learning where the robot actively participates in the learning process by asking questions that can maximize its learning gain. Semi Markovian Decision Process task representations have been used to represent the task. However, the authors have not used transition probabilities to encode the possible action sequence but they have used it as a constraint network to determine the possible sequencing of actions. The authors have experimented with two representations. One is the environment state space graph and the other is action state space graph. The algorithm used to convert between the two representation has been described in [14]. In environment state space graph, each vertex represents the environment state and each directed edge represents an action that has to be performed to transition between the two environment states. In action state space graph, each vertex is an action and the edges hold the details of the preconditions that have to be satisfied for that particular action to occur. The authors have then showcased the effectiveness of using active learning for these representative task structures.

We plan to use these approaches to build our task representation model as these models can be used in an online manner.

Most of the approaches mentioned above require information such as actions being performed in the video. The next section provides details on the state of the art for action recognition.

2.2 Extracting information from videos

2.2.1 Extracting actions performed in the videos

Action recognition is still an open problem in the field of computer vision. In action recognition, a given sequence of images have to be labeled rather than a single image. The length of this sequence can be arbitrary. Hence the complexity in action recognition comes with the presence of an extra temporal dimension.

Convolutional Neural Networks are state of the art models in the field of image analysis [30][21]. Due to their outstanding performance in the field of image analysis, a lot of work has been done to modify the CNN architecture and make it suitable for action recognition as well.

Below are the details of the various extensions made to CNN architecture to incorporate the temporal dimension:

Single stream networks

In one of the earliest approaches to using CNNs for action recognition [44], the authors have tackled the problem of dealing with the extra temporal domain by completely ignoring it. Most actions can be defined by the objects that are manipulated. Hence the authors have tried to decipher the action that is being performed by first detecting the objects that are being manipulated using a CNN and then using a language model trained on the Gigaword corpus (Graff 2003) to determine the action that is being performed. The language model calculates the probability of an action given the objects. The action with the highest probability given the detected objects is chosen as the action.

In [18], various approaches to extending the connectivity of a CNN in the time domain has been studied. The three broad connectivity patterns are

- Early fusion: Early fusion combines the data across various frame on a pixel level by modifying the filters in the first convolutional layer to account for the temporal dimension. The kernel size used is $11 \times 11 \times 3 \times T$ where T is the number of frames. Motion direction and speed can be precisely detected using this method.

- Late fusion: Late fusion uses two separate single frame networks which work on frames that are 15 time steps apart. The two networks share parameters. These two networks are then combined with fully connected layers. This method can detect global characteristics by comparing the outputs of fully connected layers for the two frames.

- Slow fusion: Slow fusion is a balance between the above mentioned methods. In slow fusion temporal information is fused throughout the network as shown

in the figure below and hence the higher level get more global information in both the spatial and temporal dimension. The model described in the paper uses a filter with temporal dimension of 4 frames in the first convolutional layer on an input clip of 10 frames. The second and third convolutional layer use a filter with a temporal dimension of 2.

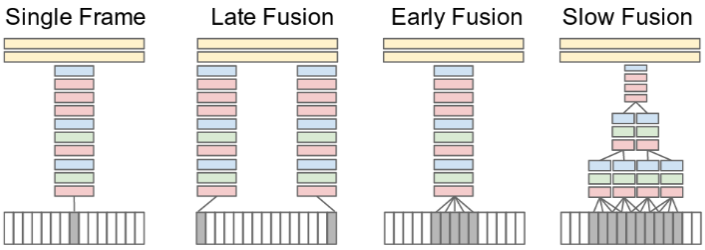

Figure 2.1: Single Stream Network [18]

For final prediction, multiple clips are sampled from the video and the prediction scores are averaged over all the sample clips. Amongst the connectivity patterns mentioned above, slow fusion performed the best. However these model were not able to capture temporal features well and hence performed significantly worse than feature based methods which were state of the art at the time.

Two stream convolutional networks:

Two stream convolutional networks [35] were introduced later for action recognition. This network consists of two separate networks called spatial stream network and temporal stream network. The spatial network learns to predict the action using a single frame whereas the temporal stream tries to predict the action by using dense optical flow of 16 frames. The two outputs are later combined by using late fusion. This method performed significantly better than the single stream deep models and the performance was comparable to the feature based method.

This method, however, requires optical flow vectors to be precomputed and stored. This model is not an end to end trainable model and hence both the models had to be trained separately. Also, since the optical flow was computed across only 16 frames the two stream model were still not able to capture long term temporal features.

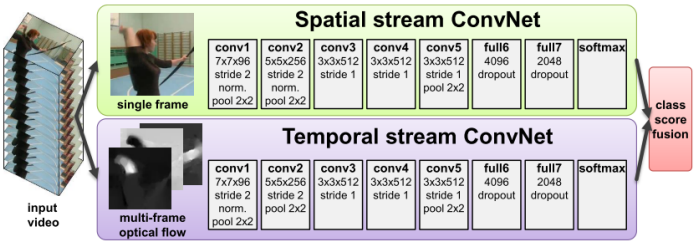

Figure 2.2: Two-stream architecture [35]

The two stream architecture described above was also not able to learn the pixel-wise correspondences between spatial features and temporal features as the outputs of the two networks were combined only at the softmax layer. In [11], the authors have argued that to differentiate few actions, it is important to map spatial features of a specific region to the corresponding region in the temporal feature map. The authors have provided the example of discriminating between brushing teeth and brushing hair actions. "If a hand moves periodically at some spatial location then the temporal network can recognize that motion, and the spatial network can recognize the location (teeth or hair) and their combination then discriminates the action [11]." Since the two stream convolutional networks fuse the value at the softmax layer, these details are lost. Hence, in this paper, the authors have experimented with various ways such as sum, max, concatenation, bi-linear fusion for combining the spatial and temporal towers spatially at a given time T. The authors also experimented with temporal fusion as well. This is done by performing 3D pooling/3D convolution+pooling over stacked spatial maps across time $t = 1...T$.

"The spatio and temporal networks are later fused at the last convolutional layer to convert it into a spatiotemporal stream by using 3D conv fusion followed by 3D pooling[11]." The final architecture proposed by the authors is in fig 2.3.

Temporal Segment networks[41] is another extension of the two stream architecture. Usually most activities span a large number of frames and with the fixed architecture like two stream network only a fixed number of frames sampled at a predefined sampling rate can be worked on. Hence the long range temporal structure is lost.

The authors have introduced a new model named Temporal Segment Network(TSN) to tackle this issue. The network architecture is the same as the base

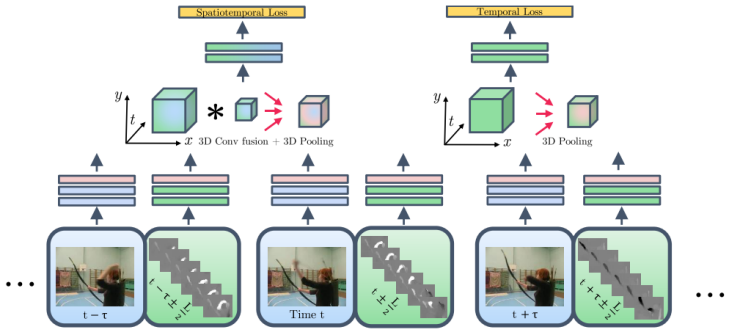

Figure 2.3: Spatiotemporal fusion ConvNet [11]

two stream model but the sampling process to generate a fixed length clip has been changed. The input video for each action is first split into 16 (the base two stream worked on clips with 16 frames) different segments. So in case one action spans for 128 frames, each segment will contain 8 frames. Once this is done, a single frame is randomly sampled from each segment and a 16 frame long clip is created which is provided as input to the network. So in TSN, the input to the network will be a clip sampled in this manner. The class scores are determined for multiple input clips. The final class scores for the input video will be average of all these values. Hence the authors provided an effective solution towards long range temporal modeling. This paper has also demonstrated the usage of techniques such as batch normalization, dropout and pretraining to avoid overfitting.

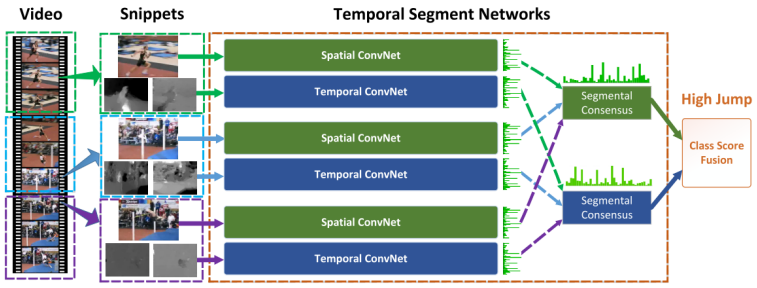

Figure 2.4: Temporal segment network [41]

The main disadvantage of the two stream networks is the necessity of precomputing optical flow for the temporal network. The authors in [45] have tried to overcome this issue, by using a CNN model to compute optical flow on the fly. The CNN

model is trained in an unsupervised learning to generate optical flow by minimizing the difference between the next frame and the frame generated by applying the predicted optical flow to the current frame. Hence the problem of generating optical flow as been posed as an image reconstruction problem. The authors have named this network as Motionnet. The Motionnet is then concatenated with the temporal network and the entire model is finetuned in an end to end manner to predict the action labels for the given set of frames. With this approach, a significant speed up of prediction was achieved while maintaining accuracy level similar to the base two stream architecture.

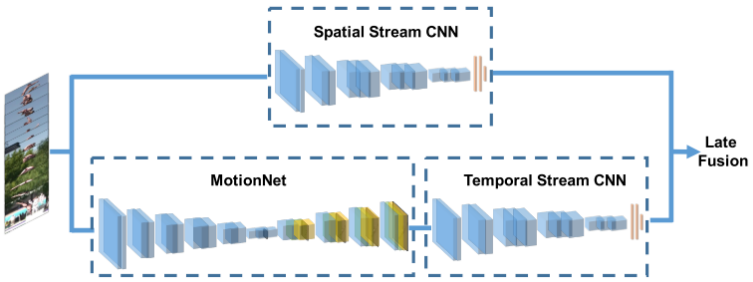

Figure 2.5: Hidden two-stream networks [45]

The two stream networks require the optical flow vectors to be precomputed and stored. This is a major bottleneck for speeding up the prediction process. Even with hidden two stream networks, the accuracy was only similar to the initial baselined two stream network architecture.

3D CNN models

3D Convolutional Neural Network models [17] create feature from both the spatial and the temporal domain, such that they capture the motion information encoded in multiple adjacent frames by performing 3D convolutions on a fixed number of frames. Each kernel in the 3D CNN model has an extra depth dimension when compared to a 2D CNN to account for the temporal changes.

In [40], the authors have shown that an 3D CNN architecture with $3 \times 3 \times 3$ convolution kernel performs best. The authors have also shown that the features learned using 3D CNN named as C3D when used with a simple linear classifier

outperforms state of the art methods on various benchmarks. The architecture that has been developed in the paper is as shown below:

Figure 2.6: C3D architecture [40]

In [7], the authors have released a new dataset named kinetics. This dataset is significantly larger than the previous baseline datasets like UCF101 and HMDB51. The authors have trained different networks such as Convnet+LSTM, two stream CNN, 3D CNN and a new architecture named Two stream Inflated 3D convnets (I3D) and have provided a comparison of the accuracy of the model. It was seen that I3D performed the best. This I3D architecture is an inflated version of two stream network. Since the kernels are just the inflated version of 2D CNN, Imagenet model weights are replicated across the temporal domain for weight initialization. They have also experimented with pretraining the models with the kinetic dataset and then performed fine tuning on UCF and HMDB51. It was seen that the Conv3D and I3D benefited significantly with pretraining. It was also seen that the 3D CNN was able to perform significantly better when provided with a larger dataset.

In [9], the authors have extended the DenseNet architecture [16], a 2D CNN architecture, to 3D with a new temporal pooling layer named Temporal Transition Layer. The described architecture is as shown in fig 2.7.

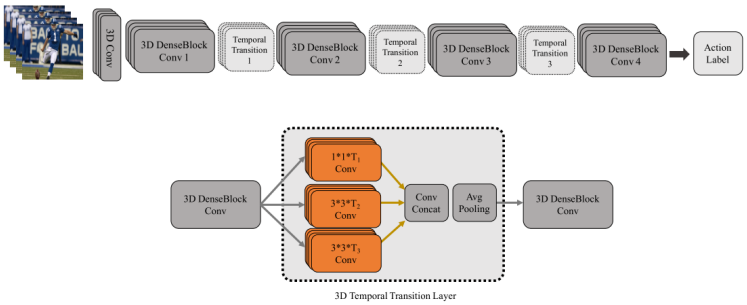

Figure 2.7: Temporal 3D ConvNet (T3D) [9]

In 3D-CNNs, the kernel depth is fixed but in general the action durations are not. Hence the authors have argued that these networks cannot capture the temporal

informations for varying lengths of actions. Inspired by GoogleNet, the authors have described a new layer named temporal transition layer. These layer consists of kernels with varying depth. The output from each of the kernel are concatenated and average pooling is performed on them. Hence this network can better model temporal information. The authors have also described an approach to achieve transfer learning between 2D architecture which is pretrained on Imagenet to a 3D CNN architecture. The architecture for knowledge transfer is described in fig 2.8.

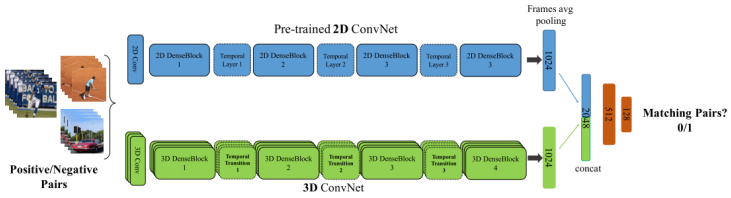

Figure 2.8: Architecture for knowledge transfer from a pre-trained 2D ConvNet to 3D ConvNet [9]

The authors argue that the representation learned over the set of frames in the 2D CNN should be similar to the representation learned by the 3D CNN. So given n frames, the frames average pooling layer will perform average pooling on the n representation for the individual frames. This is then compared with the representation that has been predicted by the 3D CNN architecture. The network is trained to reduce the difference between these two representations with the weights of the 2D CNN frozen. This has been done for stable weight initialization of the 3D CNN architecture.

When provided with more data, 3D-CNN models perform better than two stream models and are the current state of the art; however, when dealing with a smaller datasets, two stream networks perform slightly better. We have used 3D-CNN models as these models are end to end trainable and they do not require optical flow data to be precomputed and stored. Also, optical flow data is computationally expensive to generate.

2.3 Datasets

Since we are interested in recognizing cooking activities, the following datasets are applicable to us: KSCGR [34], YouCook [44], TUM Kitchen Data Set of Everyday

Manipulation Activities [38] and MP-II cooking activities dataset [31].

The KSCGR dataset consists of 5 different cooking recipes performed by 5 different people with annotations for the following actions: *breaking, mixing, baking, turning, cutting, boiling, seasoning and peeling.* A Kinect sensor mounted above the Kitchen was used to record the data. The sensor records both color images and depth images of 360 ×240 at 30fps. Each video spans from 5 to 10 minutes.

The YouCook dataset consists of 88 cooking videos from youtube performing various cooking activities such as *baking, grilling, making breakfast, making sandwich, preparing salad and cooking in general.* The set of actions that have been annotated include *stir, pick up, put down, season, flip, pour.* Frame by frame object annotations have also been provided along with the action annotations.

TUM Kitchen dataset consists of videos performing the task of setting tables in a kitchen environment. This dataset along with video sequence also provides "full-body motion capture data recorded by a markerless motion tracker, RFID tag readings and magnetic sensor reading"[38].

The MP-II cooking activity dataset consists of 65 fined grained cooking action such as *slicing, pouring, adding spice, etc* and was captured in a realistic kitchen setting. The data was recorded with a 4D View Solutions system using a Point Grey Grasshopper camera with 1624×1224 pixel resolution at 29.4fps.

We have used the MP-II cooking activity dataset for training the action recognition model and also the high level task representation as this dataset is larger than the rest and also since the actions are more finely classified.

3

Methodology

The main challenges in learning from demonstration approach to teach tasks to the robot are task segmentation and generalization. With task segmentation, the robot should be able to break the entire task into multiple subtasks and be able to recognize each of the subtasks when it occurs again. The main reasoning behind this is that the entire task would not be repeated the exact way but the subtasks will be repeated. With generalization, the robot should be able to learn more as it encounters more demonstration. The methodology used to tackle both these issues have been described in this chapter.

3.1 Task segmentation

For task segmentation, we have used 3D-CNNs to determine the action that is being performed. As described in the state of the art section, a lot of work has been done to modify the structure of CNN to perform action recognition. These models have outperformed feature based models by a huge margin. A 3D-CNN like any other 2D-CNN model consists of 4 layers: convolutional layer, activation layer, pooling layer and fully connected layer.

1. **3D convolutional layer**

   The convolution operation which is defined as the "mathematical operation on two functions (f and g) to produce a third function that expresses how

one function modifies the other"[1] is the basis of a convolutional layer. 2D convolutions are used in the field of image processing to perform various tasks such as edge detection, sharpening, blurring etc. It is mathematically defined as follows [19]:

$$f(x,y) \circledast g(x,y) = \sum_{n_1=-\infty}^{\infty} \sum_{n_2=-\infty}^{\infty} f(n_1,n_2)g(x-n_1,y-n_2) \tag{3.1}$$

The convolution operation is performed on an input image by using filters/kernels. Depending on the type of kernel used various features can be extracted from the image. The convolution of an image with sobel filter, an edge detection filter, is shown in fig 3.1. Convolution operation on image modifies the size of an image. Given an image of size $W_1 \times H_1$, the output when convolving with a square filter of size f with padding P and stride s will be[2] $W_2 \times H_2$ where $W_2 = \frac{(W_1 - f_1 + 2P)}{s} + 1$, $H_2 = \frac{(H_1 - f_2 + 2P)}{s} + 1$.

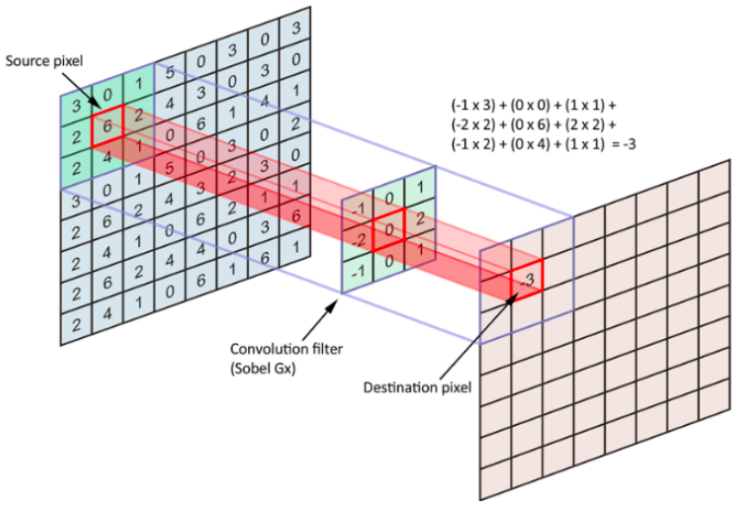

Figure 3.1: Convolution operation performed on an image with sobel kernel [3]

In CNNs, the kernels are not predefined but are learned during training of the network. The kernels learn to extract generic features from an image. In

[1] definition taken from wikipedia
[2] taken from https://leonardoaraujosantos.gitbooks.io/artificial-inteligence/content/convolution.html

2D-CNN, the convolution kernel is 3 dimensional where the third dimension is the number of channels present in the image. Hence, the convolutional layer takes a 3D volume of data as input. The filter is slided over the input image to get the values for the entire image. The output is referred to as the feature vector of the image.

With the 3D-CNN, the convolutional kernels have been expanded in the temporal dimension as well and the kernel has four dimensions $height \times width \times depth \times$ *temporal depth*, where depth is the number of channels and temporal depth is the number of frames that are considered for performing action recognition. Hence, in 3D-CNNs instead of scanning just a single image the kernels will scan a sequence of images at a time and try to extract features in both and spatial and temporal dimensions. This operation has been shown in fig 3.2.

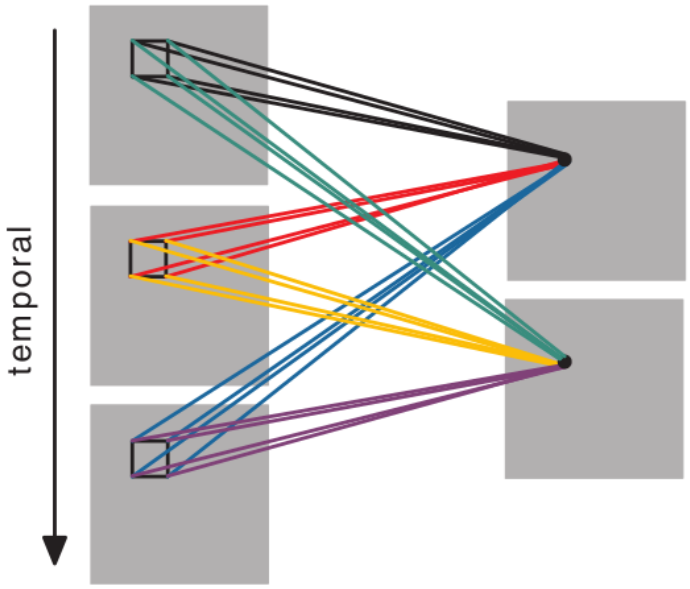

Figure 3.2: 3D convolutional kernel[17]

2. **Activation layer**

The output of the convolutional layers is passed through an activation layer. This is used to add non-linearities to the network so that the network can approximate arbitrarily complex functions. *Sigmoid, tanh and relu* activation

are generally used. *Relu* is the most common one as the convergence is much faster [20] and it avoids the vanishing gradient problem[12].

Relu is defined as follows:

$$R(x) = \begin{cases} x & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

3. **Pooling layer**

Pooling is performed to reduce the dimensionality of the data. This is done to deal with the computational complexity by reducing the size of the feature vector. It also adds translation invariance to the network. Fig 3.3 shows pooling performed with a kernel size of 2×2.

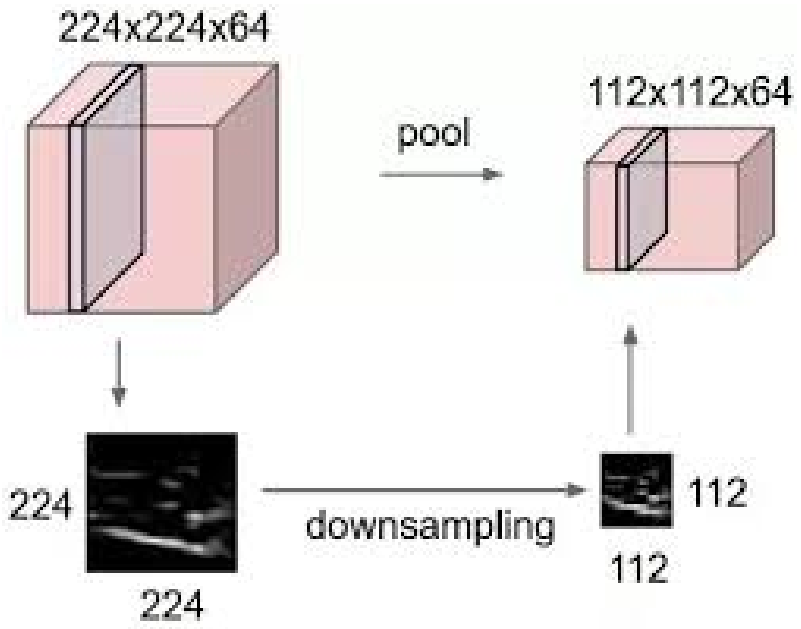

Figure 3.3: 2D max pooling operation [2]

In 3D-CNN, pooling is performed in the temporal dimension as well. Given an input of shape $W_1 \times H_1 \times D_1 \times N_1$ ($width \times height \times depth \times features$) to a pooling layer with parameters: pooling extent consisting of a tuple of 3 values ($f_1 \times f_2 \times d$) representing the values in 3 dimension and stride consisting of

a tuple of 3 values (s_1, s_2, s_3), the output shape will be[3] $W_2 \times H_2 \times D_2 \times N_1$ where $W_2 = \frac{(W_1 - f_1)}{s_1} + 1$, $H_2 = \frac{(H_1 - f_2)}{s_2} + 1$ and $D_2 = \frac{(D_1 - d)}{s_3} + 1$.

The two most common ways of performing pooling are max pooling and average pooling. Max pooling retains the max value present in the receptive field of the kernel whereas average pooling calculates the average value of all the values present in the receptive field and retains that value.

4. **Fully connected layer**

   The neurons in fully connected layer *i* is connected to every other neuron in layer *i+1*. The fully connected layer classifies the representation that has been learned in the previous layers into classes. The output of the preceding layer (convolution/pooling) is flattened to a 1 dimensional vector and then passed to the fully connected layer. The output of fully connected layers is a vector of size M, where M is the number of classes to be classified, with each value in the vector representing the probability of the sample being a specific class.

3D-CNN networks are trained using gradient-based optimization algorithms. We have used an algorithm named Adam [29] to train the network as it has shown to perform better compared to other methods such as Stochastic Gradient Descent, Stochastic gradient descent with momentum, Adagrad and RMSProp [32]. Categorical cross entropy loss function has been used. It is defined as follows[4]:

$$Loss = \sum_{c=1}^{M} y_{o,c} \times ln(p_{o,c}) \tag{3.2}$$

Here *M is the number of classes, p is the predicted probability observation o is of class c and y is a binary indicator which is defined as follows*

$$y_{o,c} = \begin{cases} 1 & \text{if } o = c \\ 0 & \text{otherwise} \end{cases}$$

It can be seen that this loss function, for a given sample, tries to maximize the probability of the correct class while minimizing the probabilities of other classes.

[3]These formulas have been taken from http://cs231n.github.io/convolutional-networks/ and adapted for 3D-CNN

[4]taken from https://ml-cheatsheet.readthedocs.io/en/latest/loss_functions.html

One of the significant problems in all deep learning methods is overfitting. Overfitting is the case where the model ends up memorizing the training data rather than determining the pattern present in them. It can be seen that the model has overfitted when the test accuracy is much lesser than the training accuracy. Convolutional neural network have shown to be more resilient towards overfitting when compared to fully connected networks. This is due to the convolutional layer where each kernel has shared weights and it tries to extract generic features from the given input. However, the fully connected layers in the CNN are still prone to overfitting [20]. Dropout [15] is used to counter this. While using dropout, random neurons are dropped with a predefined probability. This reduces co-adaptation between the neurons and leads to better generalization. Also, it was shown that batch normalization which is regularization technique when used after each convolutional layer reduces overfitting in a CNN.

3.1.1 3D CNN implementation

Our implementation follows the one described in [40], published by Facebook. The original implementation is in Caffee but we have used keras for our implementation. We have used the two models described in the paper. The first model (*m1*) has 5 convolution layers, 5 max pooling followed by 3 fully connected layers. The second model (*m2*) has 8 convolution layer, 5 max pooling followed by 3 fully connected layers.

A stride of $2 \times 2 \times 1$ ($height \times width \times temporal$) has been used in the first max pooling layer. Pooling has not been done in the temporal domain to avoid collapsing the temporal information too early. In the subsequent pooling layers a stride of $2 \times 2 \times 2$ has been used. Also, the authors in [40] have found that a $3 \times 3 \times 3$ kernel performs the best for the convolution layer. Hence, we have followed this design decision.

The number of filters in the convolution layers for *m1* are 64, 128, 256, 256 and 256 respectively and for *m2* are 64, 128, 256, 256, 512, 512, 512 and 512 respectively.

The network parameters for *m1* are as mentioned below:

Table 3.1: Network parameters for *m1*

Layer Name	**Output Dimension**	**Number of trainable parameters**
Conv1	120, 160, 16, 64	5248
Pool1	60, 80, 16, 64	0
Conv2	60, 80, 16, 128	221312
Pool2	30, 40, 8, 128	0
Conv3	30, 40, 8, 256	884992
Pool3	15, 20, 4, 256	0
Conv4	15, 20, 4, 256	1769728
Pool4	8, 10, 2, 256	0
Conv5	8, 10, 2, 256	1769728
Pool5	4, 4, 1, 256	0
Flatten	5120	0
Dense1	4096	20975616
Dense2	512	2097664
Dense3	101	51813

The network parameters for *m2* are as mentioned below:

Table 3.2: Network parameters for *m2*

Layer Name	**Output Dimension**	**Number of trainable parameters**
Conv1	120, 160, 16, 64	5248
Pool1	60, 80, 16, 64	0
Conv2	60, 80, 16, 128	221312
Pool2	30, 40, 8, 128	0
Conv3	30, 40, 8, 256	884992
Conv4	30, 40, 8, 256	1769728
Pool3	15, 20, 4, 256	0
Conv5	15, 20, 4, 512	3539456
Conv6	15, 20, 4, 512	7078400
Pool4	8, 10, 2, 512	0
Conv7	8, 10, 2, 512	7078400
Conv8	8, 10, 2, 512	7078400
Pool5	4, 5, 1, 512	0
Flatten	10240	0
Dense1	4096	41947136
Dense2	4096	16781312
Dense3	101	413797

Due to size of the datasets for action recognition, it is not possible to load the entire dataset to the RAM at once. We have used data generator feature provided in keras[1] to deal with this problem. Data generator generates batches of desired size when needed and hence the entire dataset need not be loaded at a time. The number of batches that will be present in each epoch has to be predefined.

We have followed the augmentation approach described in [41]. Our model takes 16 frames as input hence the clips for each action is first segmented into 16 chunks. Once this is done we randomly pick 1 frames from each of the 16 segments. We have also applied image augmentation such applying a random rotation between 0° to 25° to all the frames in a clip, horizontal flip and adding random gaussian noise to the frames in the clip.

3.2 Generalization/Multi-step action model

The idea for the multi-step action model has been based on task generalization section of [10]. Given a single demonstration to the robot we assume that it is only way for performing that task. So there is no constraint on the first action that is performed and for the subsequent actions all the actions preceding it are considered as temporal constraints.

Take for example, the use case of making coffee as described in chapter 1. One way of performing this task is *turn on the stove, keep vessel on the stove, pour water into the vessel, boil, pour hot water into the cup, put sugar into the cup, put coffee powder into the cup, stir.*

Given a demonstration with this action sequence, the robot assumes that this is the only way of performing the task. Hence the robot thinks that *put sugar* action can be performed only after *turn on, keep vessel on stove, pour water into the vessel, boil, pour hot water into the cup* actions have been executed. The constraint graph after first demonstration is as shown in fig 3.4

Now, given the following demonstration: *put coffee powder into the cup, put sugar into the cup, pour water into the vessel, keep vessel on the stove, turn on the stove, boil, pour hot water into the cup, stir*; the robot can learn that the actions *turn on, keep vessel on stove, pour water into the vessel, boil, put hot water into cup* need not be performed before *put sugar* action. The constraint graph after second demonstration is as shown in fig 3.5. Edges shown in red are the temporal constraints

that have been removed after the second demonstration.

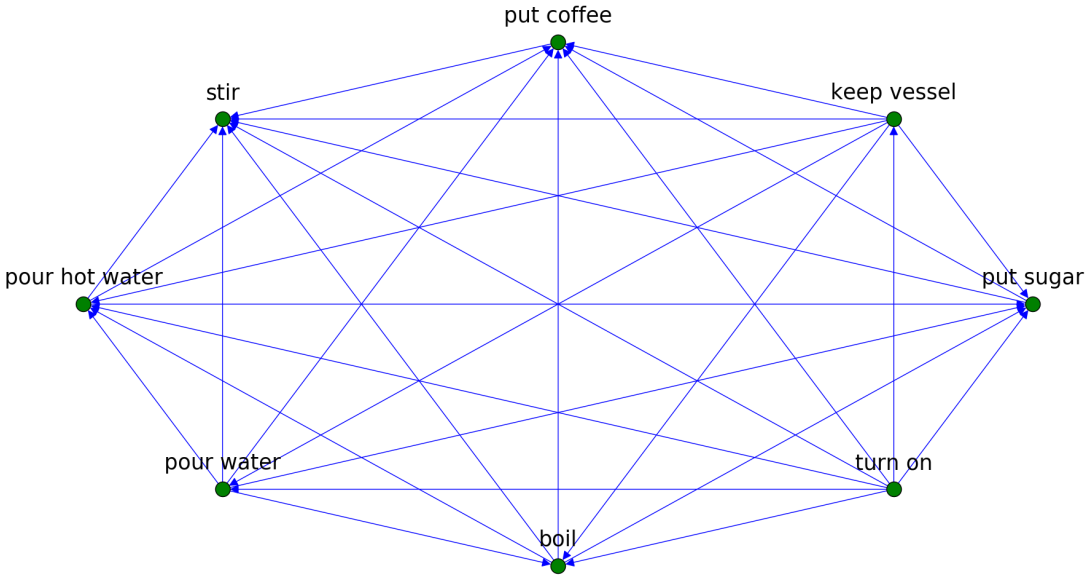

Figure 3.4: Constraint graph for coffee making use case after first demonstration

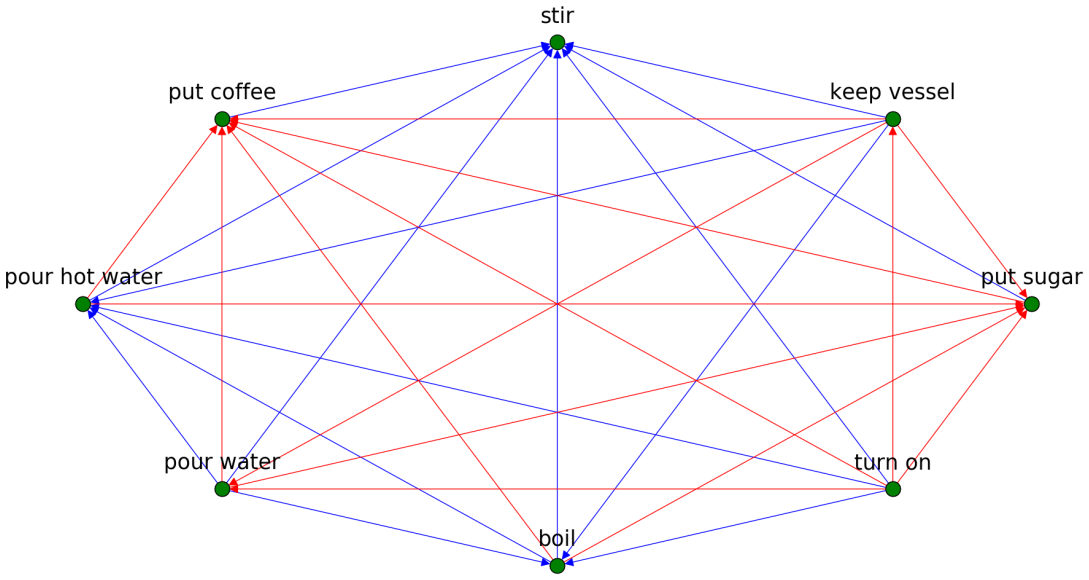

Figure 3.5: Constraint graph for coffee making use case after second demonstration

Hence, initially allowed action sequences are very restricted. Once more demonstration are provided, the model is able to learn the relevant temporal constraints on the actions that have to be performed to complete the task.

Also, the number of steps in each demonstration might vary depending on task. During the demonstration two scenarios can arise: first demonstration includes an action which is not present in the second demonstration and second demonstration includes an action which is not present in the first demonstration.

To explain these scenario better we will include an extra action of *add cream* in the coffee preparation use case.

For the first scenario we modify the second demonstration to include *add cream* action. *add cream* is performed before *stir*.

Since this is the first time we are observing this action all the actions preceding this will be taken as temporal constraints for this action. But the constraints for *stir* action is not modified due to the fact that in the previous demonstration it was seen that stir can be performed even without *add cream* action. The constraint graph is as shown in fig 3.6 and fig 3.7

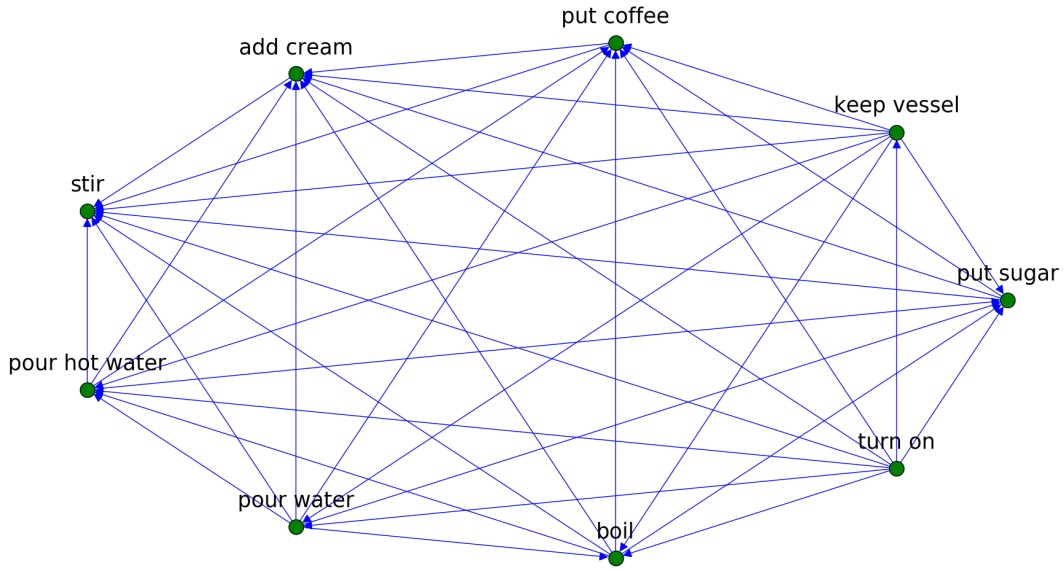

Figure 3.6: Constraint graph for coffee making use case after first demonstration

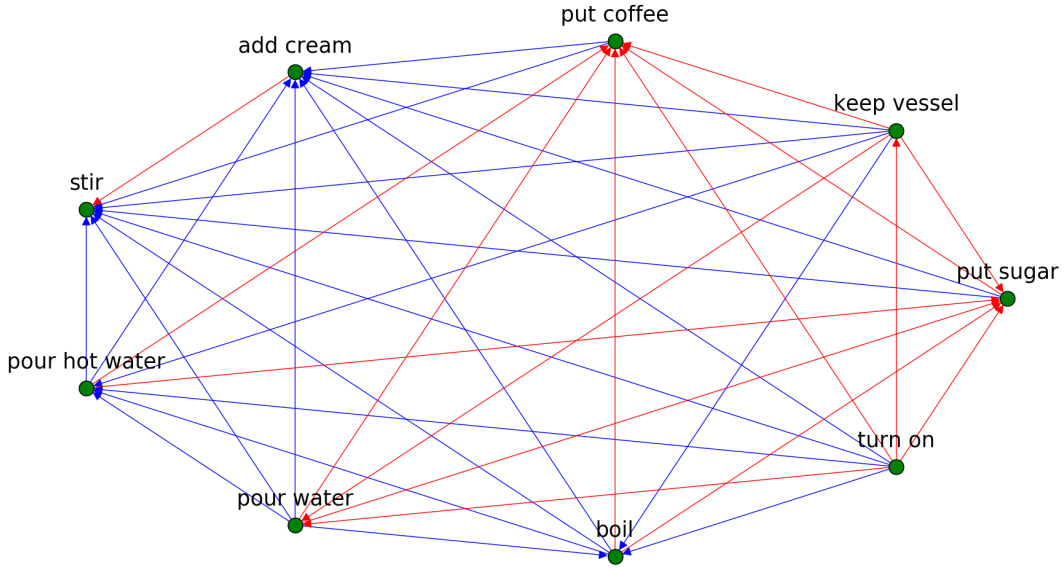

Figure 3.7: Constraint graph for coffee making use case after second demonstration

For the second scenario, we include this action in the first demonstration itself. We use the exact same steps as mentioned above but with *add cream* action being performed before stir. The constraint graph is as shown in fig 3.8

The second demonstration remains unchanged i.e. the *add cream* action is not

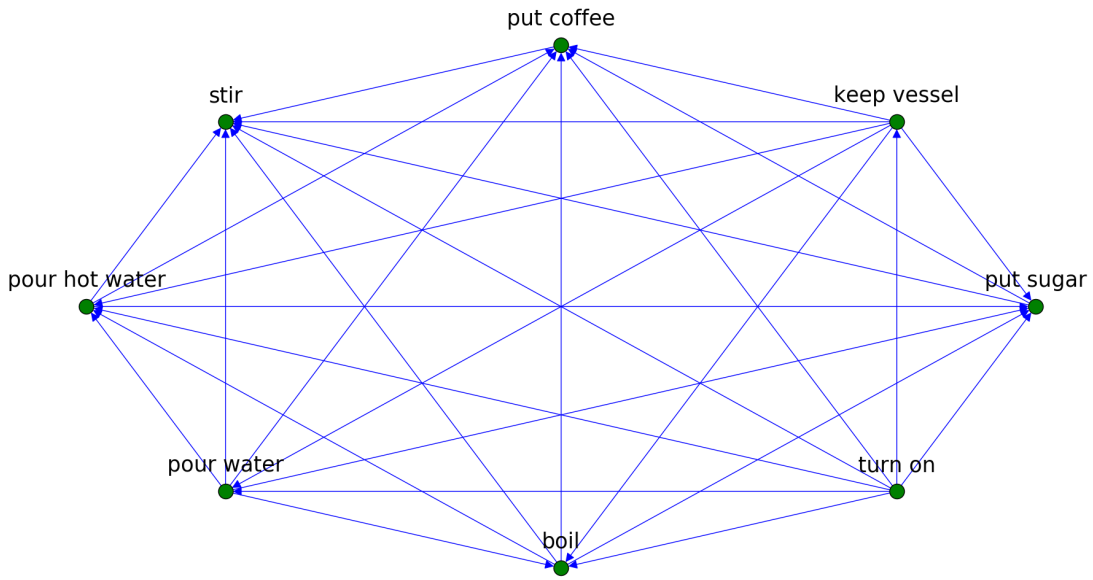

Figure 3.8: Constraint graph for coffee making use case after first demonstration

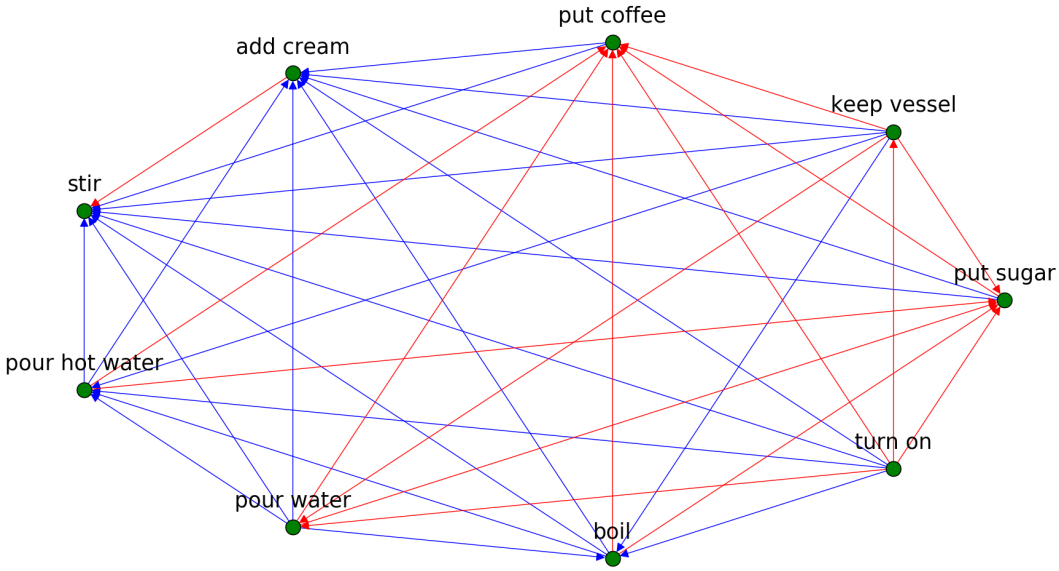

Figure 3.9: Constraint graph for coffee making use case after second demonstration

performed. Hence the constraints for this action are not modified after the second demonstration. But it was observed that *stir* action can performed successfully even without add cream action. Hence, this constraint is removed from *stir*. The constraint graph after second demonstration is as shown in fig 3.9

The formalization of the above mentioned approach is as follows: following [13], we use a graph for representing temporal constraints between actions. Let's suppose that a task is performed using n primitive actions (different parameterizations of the same primitive action are allowed), such that action A_i, $1 \leq i \leq n$ starts at time ts_i and ends at time te_i. The $n \times n$ temporal adjacency matrix T of the task has the following semantics:

$$T_{j,i} = \begin{cases} 1 & \text{if } te_i < ts_j \\ 0 & \text{otherwise} \end{cases}$$

In other words, the edge between actions j and i is activated if A_i needs to be executed before A_j.

For learning the temporal constraints of a task, we assume we have a training set T_E of m labeled executions of the task, where each training instance is a list of actions in the order in which they were performed, such that we note that different executions of the task do not necessarily have the same number of actions[5]. We denote the number of actions in execution $T_{E,k}$, $1 \leq k \leq m$ by m_k and we note that

$$n = \max_k m_k \tag{3.3}$$

If we are given T_E as a whole, we could perform offline learning of T; however, in our case, we do not assume that all m executions are provided to the learning algorithm at once, but we instead consider them in a sequential online fashion, which means that we do not in fact know the value of n in advance. Because of this, we construct T iteratively starting with $n = 0$ and following a learning procedure inspired by [42] that performs two major operations on T:

1. If an action $T_{E,(k,t)}$, $1 \leq t \leq m_k$ is not in T, we add it to T and set all $T_{E,(k,\bar{t})}$, $1 \leq \bar{t} < t$ as its constraints

2. If $T_{E,(k,t)}$ has been seen before, all entries in T that were observed as constraints in previous demonstrations, but are not in $T_{E,(k,\bar{t})}$, $1 \leq \bar{t} < t$ are removed as constraints

It can be noticed that this learning procedure results in a sparse strictly triangular

[5]For instance, in the case of packing a suitcase, we may have executions of the task that differ in length depending on the number of packed items.

matrix T.

A pseudo-code of the outlined learning procedure is given in Algorithm 1.

Algorithm 1 Temporal task constrain learning

```
1: function LEARNTEMPORALTASKCONSTRAINTS(T_E)
2:     T ← ()
3:     n ← 0
4:     for k ← 1 to m do
5:         for t ← 1 to m_k do
6:             if T_{E,(k,t)} not in T then
7:                 T ← T ∪ zero row ∪ zero column
8:                 n ← n + 1
9:                 for t̄ ← 1 to t − 1 do
10:                    i ← index of T_{E,(k,t̄)} in T
11:                    T_{n,i} ← 1
12:            else
13:                j ← index of T_{E,(k,t)} in T
14:                for i ← 1 to n do
15:                    if T_{j,i} = 1 and A_i not in T_{E,(k,t̄)}, 1 ≤ t̄ < t then
16:                        T_{j,i} ← 0
17:    return T
```

3.2.1 Action models used

The action models which we are interested in has been described below.

Table 3.3: Action models

Action name	Parameters	Description
add	x,y	add x to y
blend	x	blend the contents of x

Table 3.3 : Action models

Action name	**Parameters**	**Description**
cut_apart	x	cut apart x
cut_dice	x	cut x into dices
cut_in	x	cut inside of x
cut_off_ends	x	cut off ends of x
cut_slices	x	cut x into slices
cut_stripes	x	cut x into stripes
fill_water	x,y	fill water into x from y
grate	x	grate x
keep_on	x,y	keep x on y
mash	x,y	mash x inside y
open	x	open x
peel	x	peel x with peeler
plug_in	x	plug in extension of x into socket
plug_out	x	plug out extension of x from socket
pour	x,y	pour x into y
pull_out	x,y	pull out x from y
put_in	x,y	put x inside y
put_on	x,y	put x on y
put_on_lid	x	put on lid of x
remove	x,y	remove x from y
remove_lid	x	remove lid of x
rip_open	x	rip x open
screw_close	x	screw x close
screw_open	x	screw x open
spread	x,y	spread x on y
squeeze	x	squeeze x
stir	x	stir contents of x
strew	x,y	strew x on y
take_out	x,y	take out x from y
throw	x,y	throw x into y

Table 3.3 : Action models

Action name	**Parameters**	**Description**
turn_on	x	turn on x
turn_off	x	turn off x
wash	x	wash x with water
whisk	x	whisk x

The object recognition module required to detect the objects on which the action is being performed has not been developed with this project.

4

Evaluation and Results

We have evaluated the 3D-CNN action recognition and multi-step action model separately.

4.1 3D CNN model

We trained and evaluated the performance of the two 3D-CNN architectures described in chapter 3 with adam optimizer and loss function of categorical cross entropy on the MP-II cooking action dataset as well as the UCF-101 dataset[36] with 3 different variations of the learning rate:

- Initial learning rate of $3e^{-5}$ with a decay of 90% after every 4 epochs.
- Constant learning rate of $3e^{-5}$.
- Initial learning rate of $3e^{-4}$ with a decay of 90% after every 4 epochs

We repeated the training of the models with and without dropout.

4.1.1 Hardware setup

We perform the training of the models described in chapter 3 on the university cluster with the following hardware specifications:

Table 4.1: Hardware setup

Processor	Intel Xeon Gold 6130 (Skylake EP)
Clock speed [GHz]	2.1 (3.7 with TurboBoost)
Threads per node	64
GPU	Tesla V100-PCIE-16GB
Main memory [GB]	192 DDR4-2466 memory

4.1.2 UCF-101 dataset

UCF-101 is a benchmark dataset for action recognition. Hence, to determine if the architecture is good enough for action recognition we first validate our model on this dataset. UCF-101 contains videos of the following 5 categories: human-object interaction, body-motion only, human-human interaction, playing musical instruments and sports. It consists of 13320 videos with an average of around 100 videos per action. All the videos are recorded at a fixed resolution of 320×240 resolution at a rate of 30 fps.

All the actions clips in the dataset fall into 25 different groups. Each group shares certain features such as actors or background. Sharing the clips from same group between train and test would result in bias in the resulting accuracies. The authors have themselves provided test/train splits to make sure that clips from same group are not shared in the test/train split [1]. We have used this test/train split to split the dataset, train and test our model.

The number of frames per action span from 27 to 1775. The sampling methodology described in 3.1.1 has been used to extract a clip of 16 frames for each action. Each frame is down sampled by a factor of 2 and a batch size of 24 is used due to memory restrictions. The training of the network is done for 24 epochs.

Results on the UCF-101 dataset

The train and test accuracy for the two models described in chapter 3 for three different learning rates in UCF101 dataset has been described in table 4.2. We could see that the network was not able to converge for learning rate greater than $3e^{-5}$.

[1] http://crcv.ucf.edu/data/UCF101/UCF101TrainTestSplits-RecognitionTask.zip

Table 4.2: Accuracy for 3D-CNN on UCF101

Model	Learning rate	Train accuracy	Test accuracy
m1	$3e^{-5}$ with decay of 90% after 4 epochs	57.70	24.52
	$3e^{-5}$	100	36.09
	$3e^{-4}$ with decay of 90% after 4 epochs	1.25	1.10
m2	$3e^{-5}$ with decay of 90% after 4 epochs	73.06	28.46
	$3e^{-5}$	100	36.54
	$3e^{-4}$ with decay of 90% after 4 epochs	2.46	1.47

Both the models were overfitting to the provided data. Hence, we used dropout with probability of 0.5 and 0.2 on *m1* as it is smaller model and is less prone to overfitting. The train and test accuracy of model *m1* with dropout has been described in table 4.3. It was seen that the network was not able to learn when dropout probability of 0.5 was used. With dropout probability of 0.2, the network performed better but was still overfitting to the data. We then used batch normalization along with dropout of 0.2 on *m1* which provided the best test accuracy of 48.19% with learning rate $3e^{-5}$.

Table 4.3: Accuracy of model *m1* on UCF101 with dropout

Learning rate	Dropout probability	Train accuracy	Test accuracy
$3e^{-5}$	0.5	2.57	1.92
$3e^{-5}$ with decay of 90% after 4 epochs	0.2	34.87	20.94
$3e^{-5}$	0.2	98.76	38.78

4.1.3 MP-II cooking activities dataset

The MP-II cooking action dataset consists of 64 fine grained cooking actions. All the actions which cannot be classified as any of the 64 fine grained cooking actions have been put under the category of background activity. The MP-II dataset is biased towards specific actions. To deal with this issue, we have performed data augmentation as described in section 3.1.1 so that the number of clips present for each action is at least 80. The bar plot with the number of video clips without and with augmentation excluding background activity have been shown in fig 4.1 and fig 4.2 respectively.

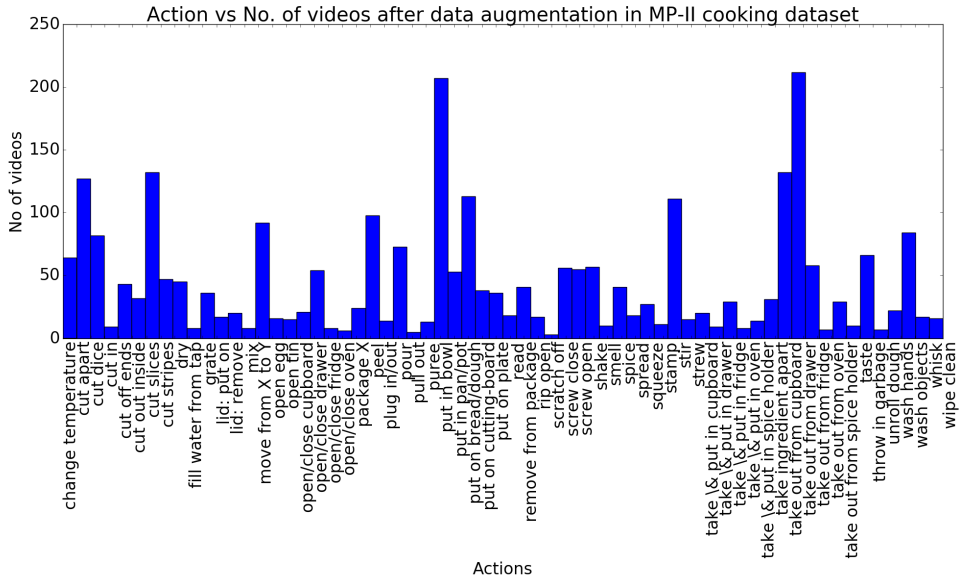

Figure 4.1: MP-II dataset (excluding background activity) without data augmentation

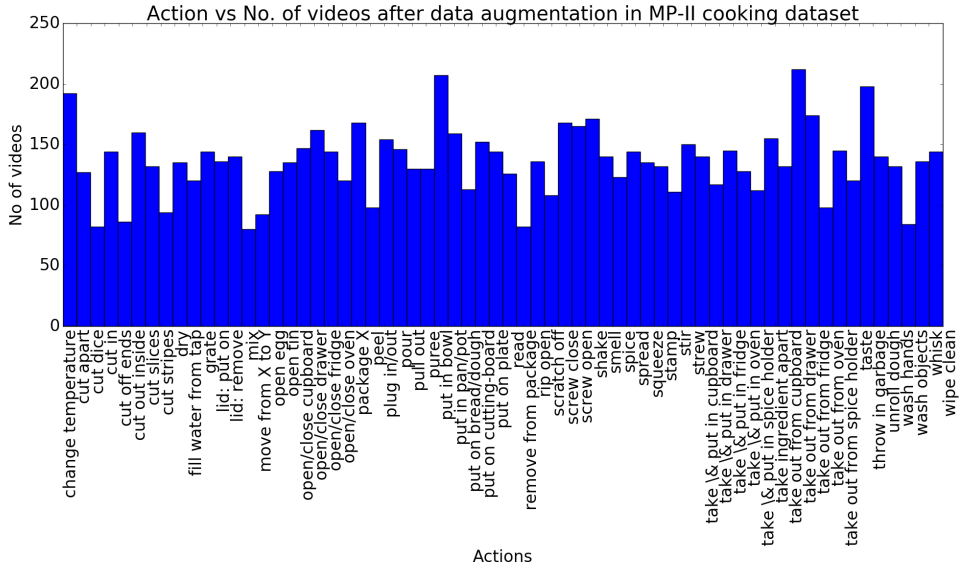

Figure 4.2: MP-II dataset with data augmentation

The number of frames per action span from 9 to 3800. The sampling procedure described in section 3.1.1 has been used to extract a clip of 16 frames for each action. The videos in the MP-II dataset have a resolution of 1624×1224. These have been down sampled by 8 and a batch size of 24 has been used due to memory restrictions. We have split the MP-II dataset in ratio of 70:30 for train and test. The training is done for 8 epochs.

Results on the MP-II cooking activities dataset

In the MP-II dataset, all the actions which cannot be classified as any of the 64 fine grained cooking actions have been put under the category of background activity. The dataset is also heavily biased towards this particular action. When we trained the C3D model including the background activity the training accuracy was around 29% and the testing accuracy was around 23%. This accuracy was achieved with the model predicting every activity as background activity and due to the bias of the dataset it looked like a decent value.

We retrained the model by removing the background activity and the accuracy dropped to 10% during training and around 2% on test set with both the models. We did see an improvement of 10% in the test accuracy when fine tuning the weights of UCF-101 dataset to MP-II cooking action dataset. The train and test accuracy for the two models have been described in table 4.4.

Table 4.4: Accuracy for 3D-CNN on MP-II pretrained on UCF101

Model	Learning parameter	Dropout	Train accuracy	Test accuracy
m1	$3e^{-5}$ with decay of 90% after 4 epochs	No	14.06	12.22
m2	$3e^{-5}$ with decay of 90% after 4 epochs	No	25.80	12.23

4.1.4 Inference of the results for 3D-CNN

From the experiments performed it was seen that the 3D-CNN was not able to learn the cooking action in the MP-II activities dataset. This is mainly due to high bias present in the dataset and not due to the 3D-CNN model as it was seen that the model was able to learn the actions present in UCF-101 dataset. Due to the bad

performance of the 3D-CNN model on MP-II dataset we did not perform any testing on the robot.

4.2 Multi-step action model

We evaluate the action representation learning model that has been developed on the following three use cases:

- Coffee making
- Mashed potato preparation
- Sandwich preparation

All the demonstrations for the above mentioned use cases are from MP-II cooking action dataset. The action sequences provided have been preprocessed to remove *Background activity* action and have been parameterized with the objects on which the actions are being performed.

4.2.1 Use case 1: coffee making

Most of the demonstrations for this use case involve the following actions: taking out items such as cups or coffee mix or chocolate from a cupboard, taking out items such as grater or vessel from a drawer, turning on/off the stove, pouring and stirring.

The MP-II dataset has four videos with slight difference in making coffee:

- The first demonstration is of making coffee with chocolate.
- The second demonstration is of making coffee.
- The third demonstration is one in which the coffee is served in a cup placed on a saucer.
- The fourth demonstration is one in which the chocolate is strewed on top of coffee.

We use this use case to describe the basic working of our model as there is only slight variation between demonstrations. With this use case, we try to show that the model is able to generalize well given multiple demonstrations.

Results for use case 1

For each demonstration we have provided details of the action sequence that has been performed and also the new actions that are seen in that particular demonstration[2] which were not seen in any of the previous demonstrations.

For each new action in the demonstration, all actions preceding it are taken as preconditions and this can be verified in the constraint graph as well. Significant differences between the demonstration have been explained in the *inference of results section* with reference to the constraint graph.

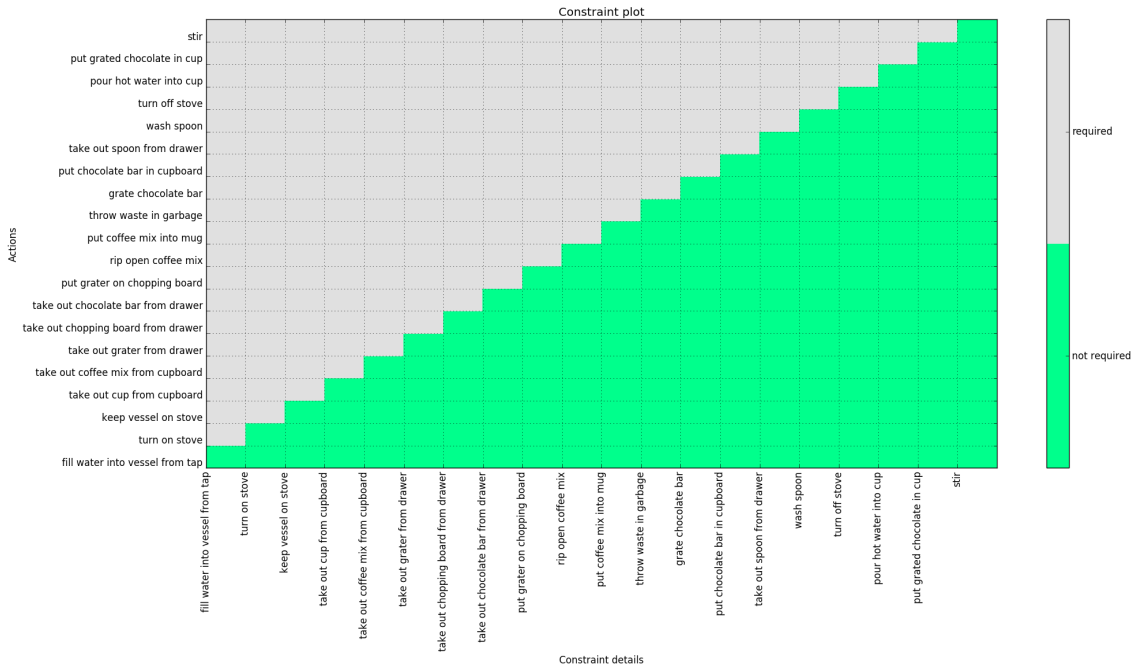

Figure 4.3: Constraint graph for use case 1 after first demonstration

Actions performed in the first demonstration [3]: *fill_water(vessel, tap), turn_on(stove), keep(vessel, stove), take_out(cup, cupboard), take_out(coffee mix, cupboard), take_out(grater, drawer), take_out(chopping board, drawer), take_out(chocolate*

[2]excluding the first demonstration

[3]The order of actions in the demonstration follows the order mentioned in the list for all the demonstrations

bar, drawer), put_on(grater, chopping board), rip_open(coffee mix), put_in(coffee mix, mug), throw(waste, garbage), grate(chocolate bar), put_in(chocolate bar, cupboard), take_out(spoon, drawer), wash(spoon), turn_off(stove), pour_in(hot water, cup), put_in(grated chocolate, cup), stir(mug).

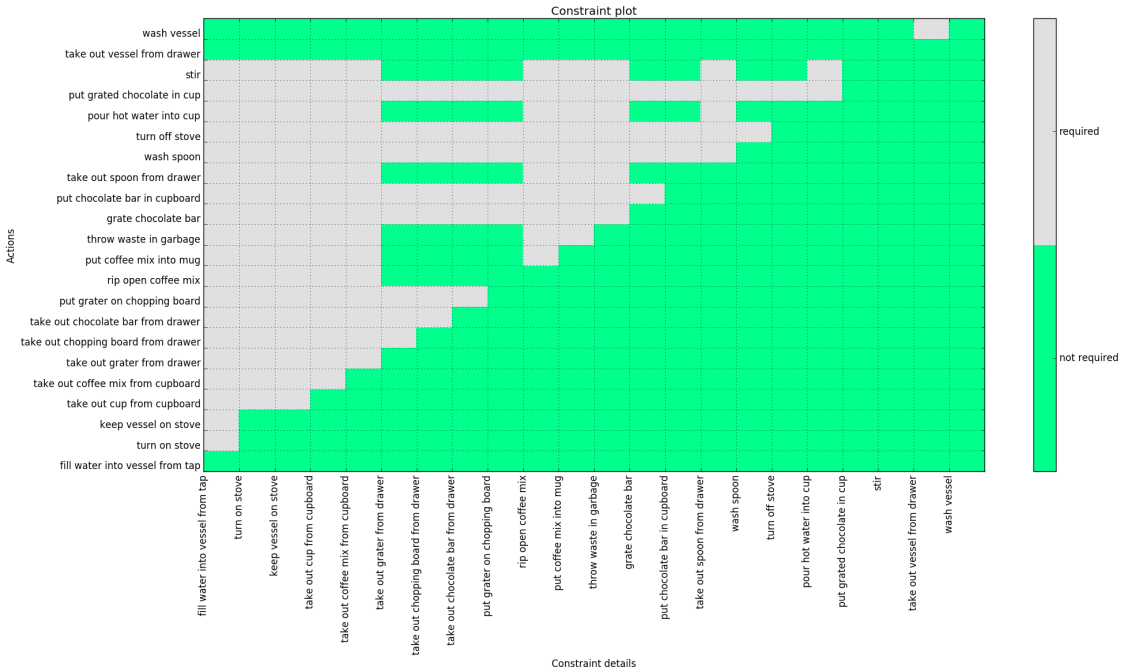

Figure 4.4: Constraint graph for use case 1 after second demonstration

Actions performed in the second demonstration: *take_out(vessel, drawer), wash(vessel), fill_water(vessel, tap), keep_on(vessel, stove), turn_on(stove), take_out(cup, cupboard), take_out(coffee mix, cupboard), rip_open(coffee mix), put_in(coffee mix, mug), throw(waste, garbage), take_out(spoon, drawer), pour_in(hot water, mug), stir(mug).*

New actions: *take_out(vessel, drawer), wash(vessel).*

Actions performed in the third demonstration: *take_out(vessel, drawer), fill_water(vessel, tap), keep_on(vessel, stove), turn_on(stove), take_out(coffee mix packs, cupboard), take_out(cup, cupboard), take_out(saucer, cupboard), take_out(scissor, drawer), take_out(spoon, cupboard), remove(pack, coffee mix pack), put_in(remaining*

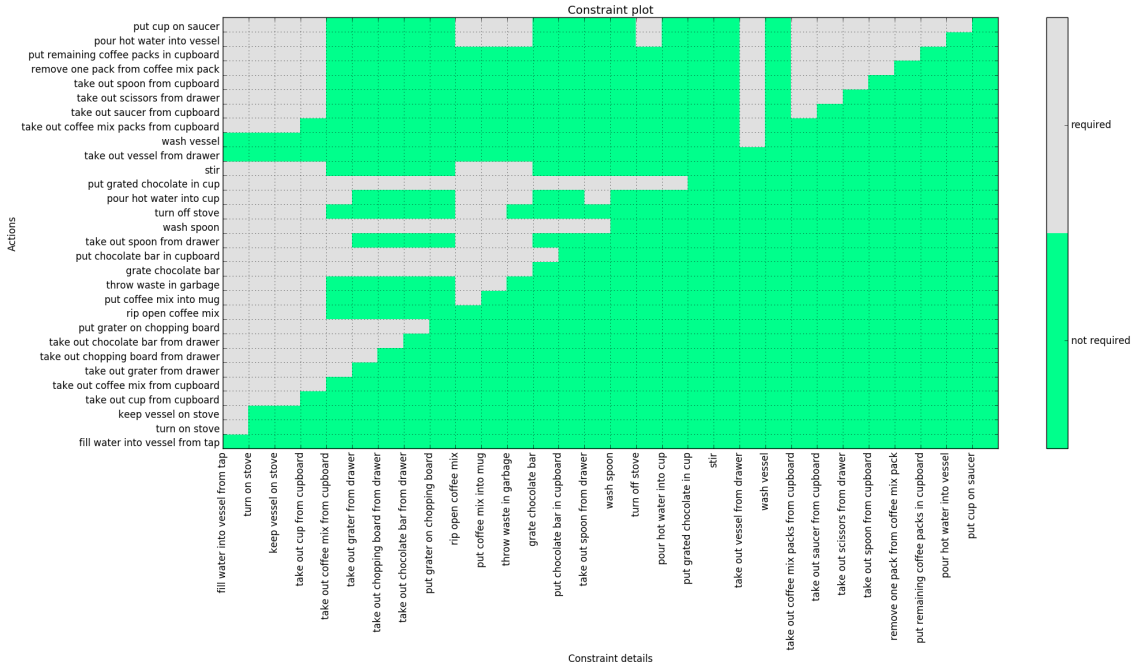

Figure 4.5: Constraint graph for use case 1 after third demonstration

coffee packs, cupboard), rip_open(coffee mix), put_in(coffee mix, mug), turn_off(stove), throw(waste, garbage), pour_in(hot water, mug), put_on(mug, saucer), stir(mug).

New actions: *take_out(saucer, cupboard), take_out(scissor, drawer), take_out(spoon, cupboard), remove(pack, coffee mix pack), put_on(mug, saucer).*

Actions performed in the fourth demonstration: *take_out(coffee mix packs, cupboard), remove(pack, coffee mix pack), take_out(vessel, drawer), fill_water(vessel, tap), keep_on(vessel, stove), turn_on(stove), take_out(cup, cupboard), take_out(saucer, cupboard), take_out(chocolate bar, cupboard), take_out(grater, drawer), grate(chocolate bar), turn_off(stove), rip_open(coffee mix), put_in(coffee mix, mug), pour(hot water, vessel), take_out(spoon, drawer), stir(mug), strew(grated chocolate, mug).*

New actions: *take_out(chocolate bar, cupboard), strew(grated chocolate, mug).*

Figure 4.6: Constraint graph for use case 1 after fourth demonstration

Inference of the results

As seen in fig 4.3, when only one demonstration is provided the model assumes that it is the only way to perform the task. Hence, the constraint for each action will be all the actions preceding it. In the second demonstration, the demonstrator has not added chocolate to coffee. Hence, all actions related to adding chocolate such as *take_out(chocolate, fridge), remove(grater, drawer), grate(chocolate) and add(chocolate, coffee)* have been removed for *stir(mug)* as these actions are not required as they have not been performed in the second demonstration. The other actions in the second demonstration were same as the first demonstration but have been performed in a different order. We can see in fig 4.4 that temporal constraints are relaxed for *throw(waste, garbage), put_in(coffee mix, mug), rip_open(coffee mix), take_out(spoon, drawer)* actions. In the third demonstration the coffee has been served in a coffee mug served on a saucer. Hence, the temporal constraints related to

actions such as *take_out(saucer, cupboard), take_out(scissor, drawer), take_out(spoon, cupboard), remove(pack, coffee mix pack), put_on(mug, saucer)* have been added as these actions are seen for the first time. Similar inference can be drawn fourth demonstration as well where *strew(grated chocolate, mug)* is a new action that is performed.

4.2.2 Use case 2: mashed potato preparation

Most of the demonstrations for this use case involve the following actions: taking out items such as a bowl or a masher from a drawer, taking items such as potatoes or milk or butter from a fridge, peeling, mashing potatoes, stirring, removing salt/pepper shaker from a spice holder, and adding salt/pepper.

The MP-II dataset has four videos with slight difference in preparing mashed potatoes:

- The first demonstration is of mashed potato with salt served with tomatoes on top.
- The third demonstration is of mashed potato with salt.
- The second and fourth demonstration is of mashed potato with salt and spices.

In this use case, most of the demonstrations have similar actions but when compared to the previous use case this task is more complicated with close to 40 actions that has to be performed to be able to complete it. Here, we mainly check if the temporal constraints are relaxed given various combinations of the given actions required to complete the task.

Results for use case 2

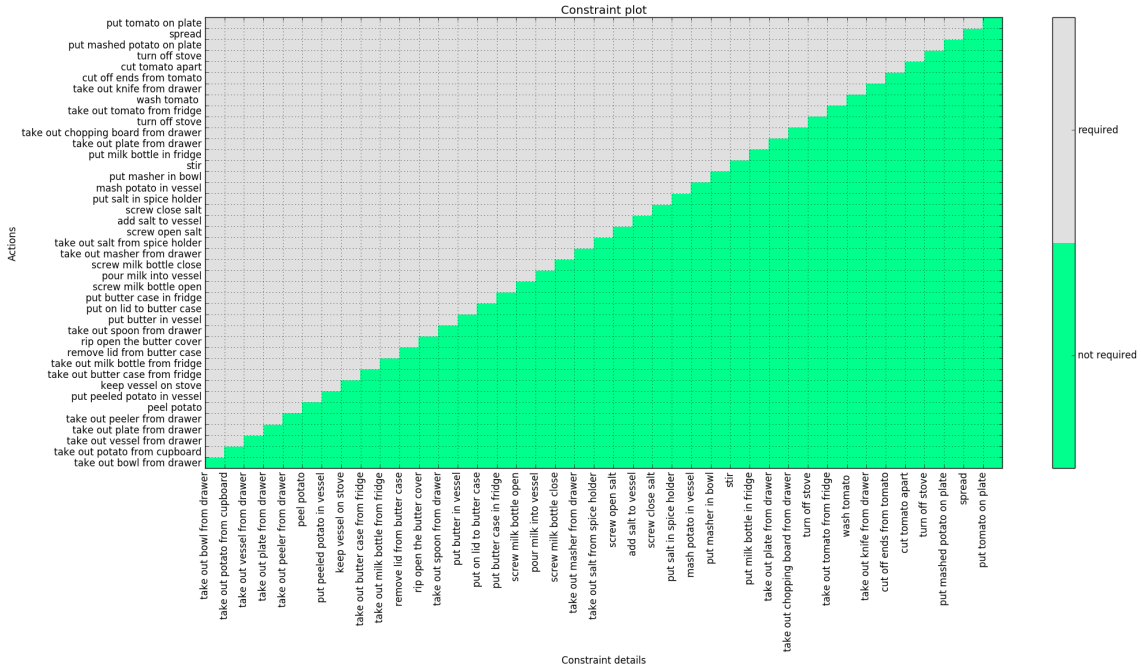

Figure 4.7: Constraint graph for use case 2 after first demonstration

Actions performed in the first demonstration: *take_out(bowl, drawer), take_out(potato, cupboard), take_out(vessel, drawer), take_out(plate, drawer), take_out(peeler, drawer), peel(potato), put_in(peeled potato, vessel), keep_on(vessel, stove), take_out(butter case, fridge), take_out(milk bottle, fridge), remove_lid(butter case), rip_opens(butter), take_out(spoon, drawer), put_in(butter, vessel), put_on_lid(butter case), put_in(butter case, fridge), screw_open(milk bottle), pour(milk, vessel), screw_close(milk), take_out(masher, drawer), take_out(salt, spice holder), screw_open(salt), add(salt, vessel), screw_close(salt), put_in(salt, spice holder), mash_in(potato, vessel), put_in(masher, bowl), stir(vessel), put_in(milk bottle, fridge), take_out(plate, drawer), take_out(chopping board, drawer), turn_on(stove), take_out(tomato, fridge), wash(tomato), take_out(knife, drawer), cut_off_ends(tomato), cut_apart(tomato apart), turn_off(stove), put_on(mashed potato, plate), spread(mashed potato), put_on(tomato, plate).*

Actions performed in the second demonstration: *take_out(vessel, drawer),*

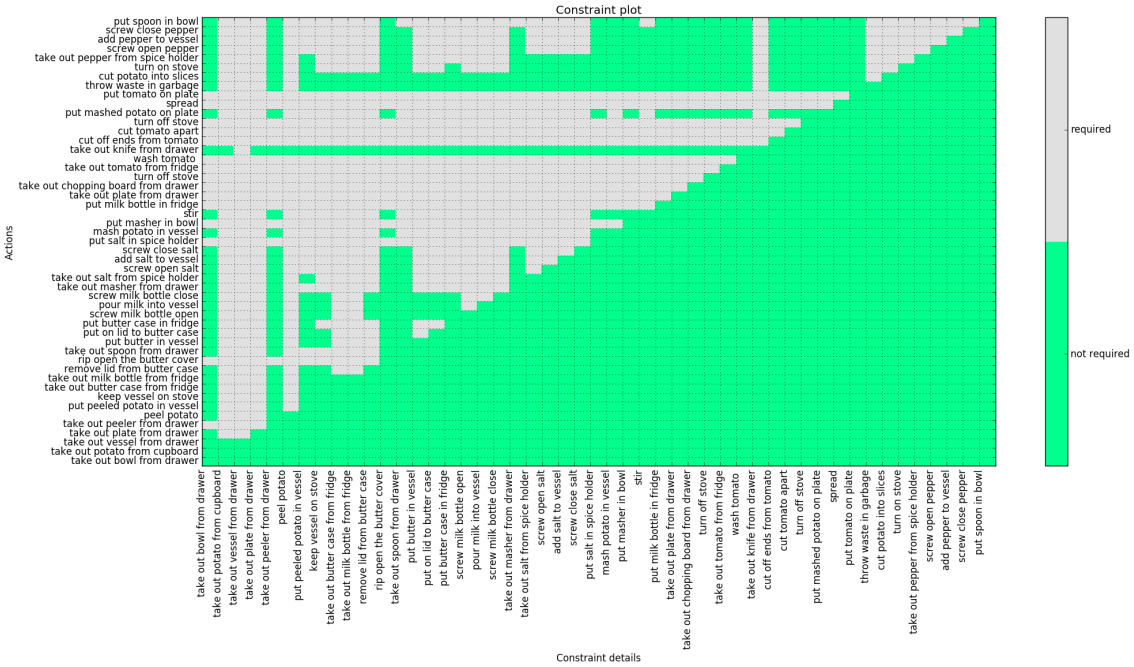

Figure 4.8: Constraint graph for use case 2 after second demonstration

take_out(knife, drawer), take_out(potato, cupboard), take_out(plate, drawer), peel(potato), throw(waste, garbage), cut_into_slices(potato), take_out(milk bottle, fridge), take_out(butter case, fridge), screw_open(milk bottle), pour_in(milk, vessel), screw_close(milk bottle), remove_lid(butter case), put_in(butter, vessel), put_on_lid(butter case), keep_vessel(stove), turn_on(stove), put_in(butter case, fridge), take_out(pepper, spice holder), take_out(salt, spice holder), put_in(peeled potato, vessel), screw_open(salt), add(salt, vessel), screw_close(salt), screw_open(pepper), add(pepper, vessel), screw_close(pepper), take_out(masher, drawer), take_out(spoon, drawer), stir(vessel), put_in(spoon, bowl), mash_in(potato, vessel), put_on(mashed potato, plate).

New actions: *throw(waste, garbage), cut_into_slices(potato), take_out(pepper, spice holder), screw_open(pepper), add(pepper, vessel), screw_close(pepper), put_in(spoon, bowl).*

Actions performed in the third demonstration: *take_out(masher, drawer), take_out(vessel, drawer), take_out(chopping board, drawer), take_out(potato, cup-*

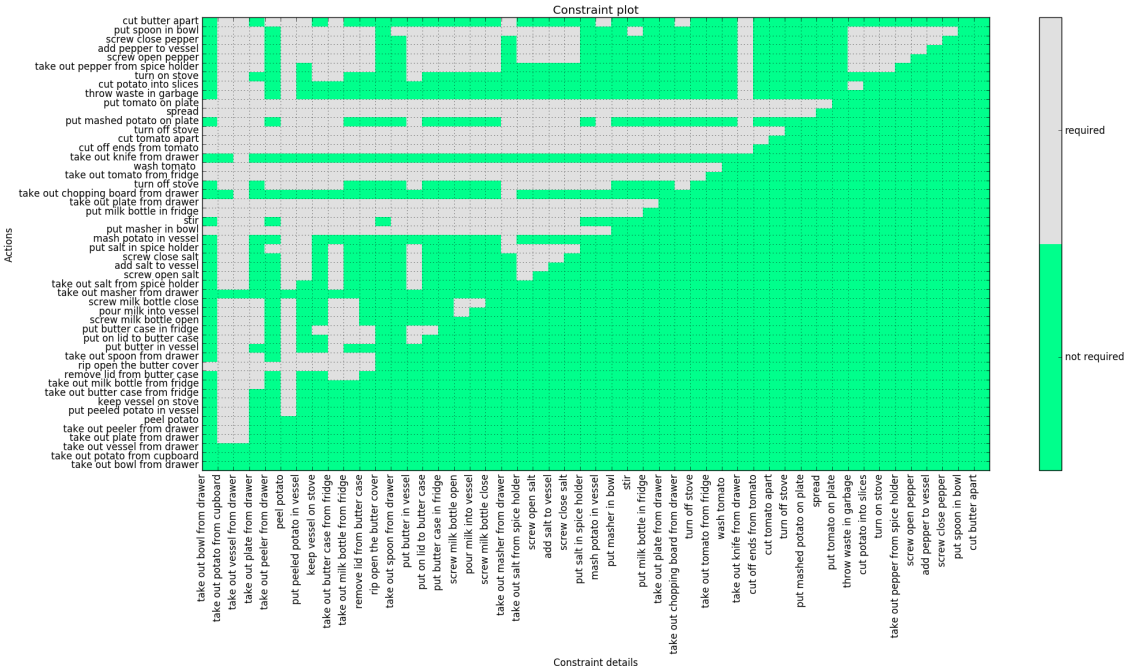

Figure 4.9: Constraint graph for use case 2 after third demonstration

board), take_out(peeler, drawer), peel(potato), put_in(peeled potato, vessel), mash_in(potato, vessel), take_out(butter case, fridge), take_out(knife, drawer), cut_apart(butter), put_in(butter), take_out(salt, spice holder), screw_open(salt), add(salt, vessel), screw_close(salt), put_in(salt, spice holder), keep_on(vessel, stove), turn_on(stove), turn_off(stove), take_out(plate, drawer), put_on(mashed potato, plate).

New actions: *cut_apart(butter)*

Actions performed in the fourth demonstration: *take_out(masher, drawer), take_out(vessel, drawer), take_out(chopping board, drawer), take_out(potato, cupboard), take_out(peeler, drawer), peel(potato), put_in(peeled potato, vessel), mash_in(potato, vessel), take_out(butter case, from fridge), take_out(knife, drawer), cut_apart(butter), put_in(butter, vessel), put_in(butter case, fridge), take_out(salt, spice holder), screw_open(salt), add(salt, vessel), screw_close(salt), put_in(salt, spice holder), keep_on (vessel, stove), turn_on(stove), turn_off(stove), take_out(plate, drawer), put_on(mashed potato, plate).*

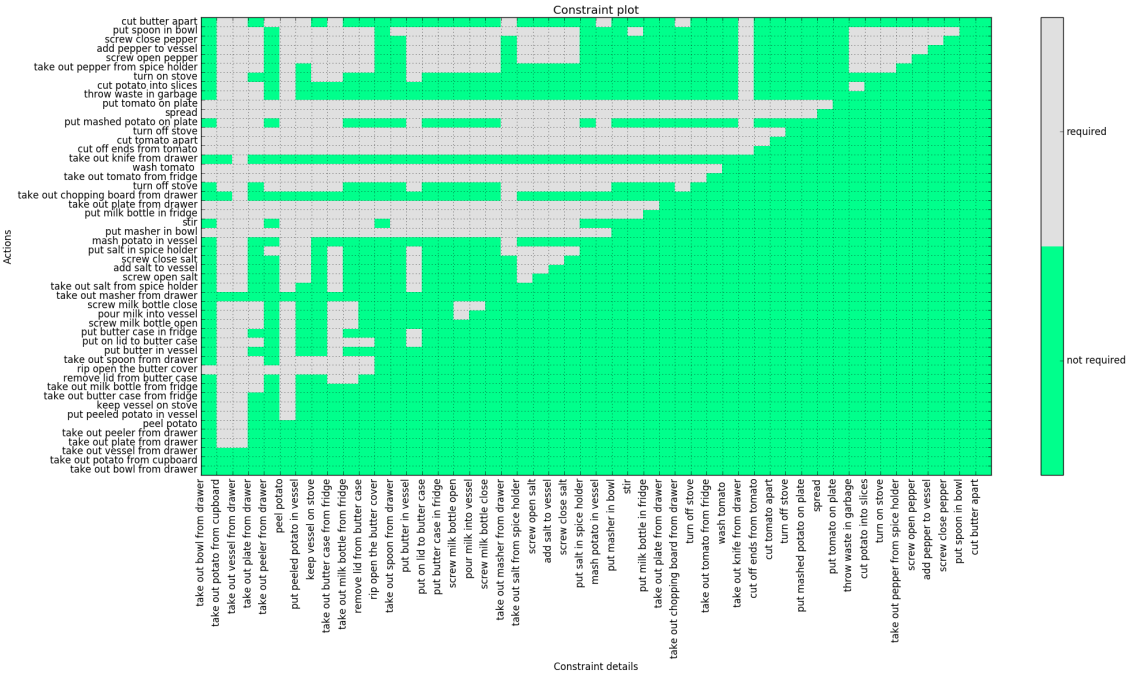

Figure 4.10: Constraint graph for use case 2 after fourth demonstration

New actions: *None*

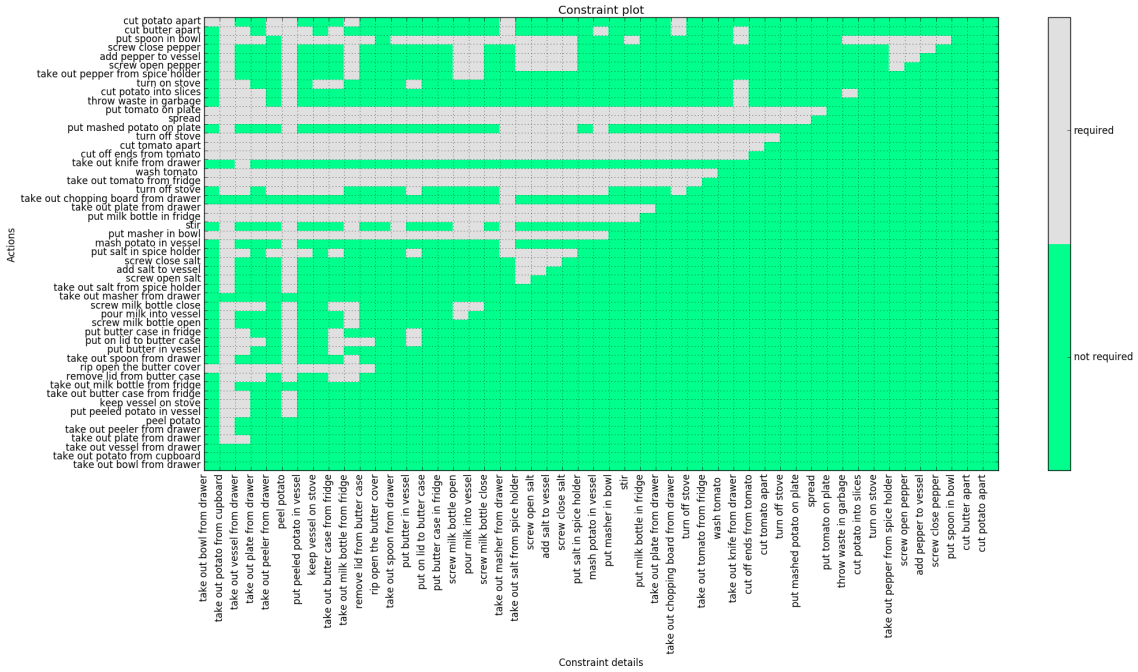

Figure 4.11: Constraint graph for use case 2 after fifth demonstration

Actions performed in the fifth demonstration: *take_out(potato, cupboard), take_out(bowl, drawer), take_out(masher, drawer), take_out(peeler, drawer), take_out(milk bottle, fridge), take_out(chopping board, drawer), peel(potato), cut_apart(potato), screw_open(milk bottle), pour_in(milk, vessel), mash_in(potato, vessel), take_out(spoon, drawer), stir(vessel), take_out(salt, spice holder), take_out(pepper, spice holder), screw_open(salt), add(salt, vessel), screw_close(salt), screw_open(pepper), add(pepper, vessel), screw_close(pepper), put_on(mashed potato, plate).*

New actions: *None*

Inference of the results

We can see in fig 4.11 that the temporal constraints were relaxed for almost all the actions except *take_out(tomato, fridge), wash(tomato), put_on(tomato,plate), cut_off_ends(tomato), cut_apart(tomato apart)* as these are related to spreading tomato on top of the mashed potato and is performed only in the first demonstration.

4.2.3 Use case 3: sandwich preparation

Most of the demonstrations for this use case involve the following actions: taking out items such as jam or bread or peanut butter from a fridge, taking out items such as knife or spoon from a drawer and spreading jam or peanut butter or butter on a piece of bread.

The MP-II dataset has three videos of preparing different sandwiches:

- The first demonstration is of peanut butter and jelly sandwich.
- The second demonstration is of peanut butter sandwich
- The third demonstration is of vegetable sandwich with cheese, butter and tomatoes.

We can see that the first two demonstrations are almost the same with the first one having more steps than the second. Hence we show that even in this case our model works fine. i.e. the constraints for the action present in the first demonstration but not in second are not modified. The third demonstrations varies significantly and we try to show that our model generalizes well even when there is significant variations in the given demonstrations.

Results for use case 3

In this use case, there is a significant difference in action that are performed in various demonstrations.

Actions performed in the first demonstration: *take_out(chopping board, drawer), take_out(knife, drawer), take_out(bread, cupboard), take_out(jam, cupboard), take_out(peanut butter, cupboard), take_out(spoon, drawer), remove(bread, package), cut_slices(bread), put_in(bread, package), screw_open(jam), put_on(jam, bread), spread_on(jam, bread), screw_close(jam), take_out(spoon, drawer), screw_open(peanut butter), put_on(peanut butter, bread), screw_close(peanut butter), put_on(one bread piece, one bread piece*

Actions performed in the second demonstration: *take_out(plate, cupboard), take_out(butter case, fridge), take_out(bread, cupboard), take_out(knife, drawer),*

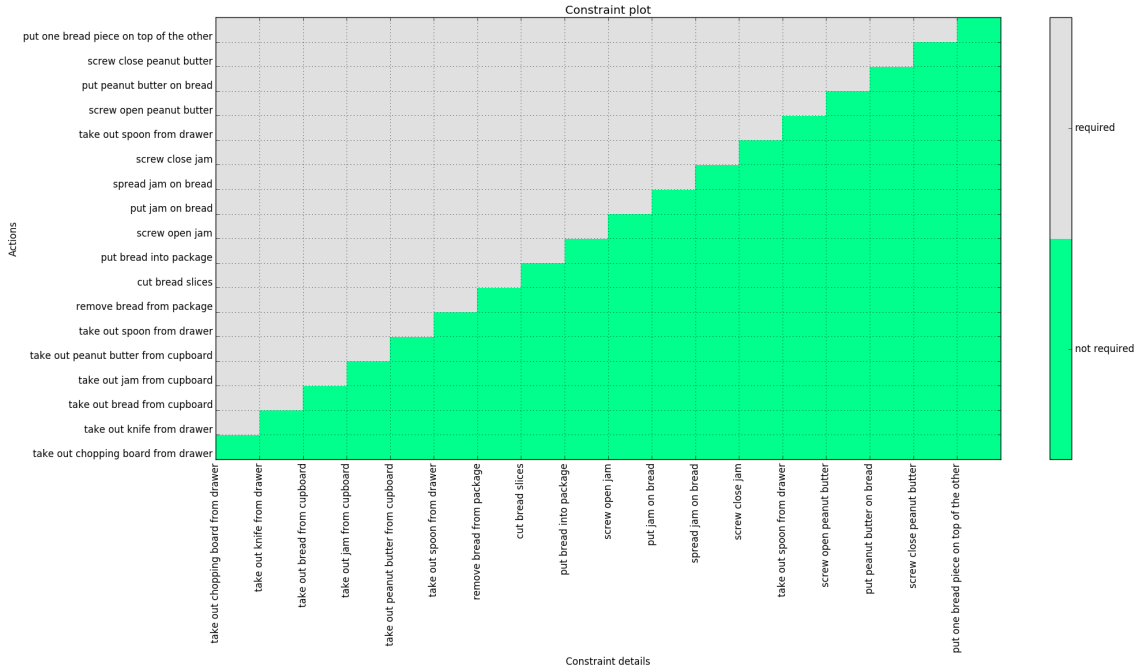

Figure 4.12: Constraint graph for use case 3 after first demonstration

take_out(chopping board, drawer), remove(bread package), cut_slices(bread), put_on(bread slices, plate), take_out(butter knife, drawer), remove_lid(butter case), spread(butter), screw_open(peanut spread), stir(peanut spread), spread(peanut spread, bread), screw_close(peanut spread), put(one bread piece, one bread piece).

New actions: *take_out(plate, cupboard), take_out(butter case, fridge), remove_lid(butter case), spread(butter), screw_open(peanut spread), stir(peanut spread), spread(peanut spread, bread), screw_close(peanut spread).*

Actions performed in the third demonstration: *take_out(chopping board, drawer), take_out(knife, drawer), take_out(bread, cupboard), take_out(butter case, fridge), take_out(plate, cupboard), remove(bread, package), cut_slices(bread slices), put_on(bread slices, plate), take_out(knife, drawer), remove(cheese, package), cut_slices(cheese), pack(cheese), take_out(butter knife, drawer), remove_lid(butter case), take_out(tomato, fridge), wash(tomato), take_out(knife, drawer), cut_apart(tomato apart), cut_slices(tomato), put_on(butter, bread), spread(butter, bread), cut_stripes(cheese),*

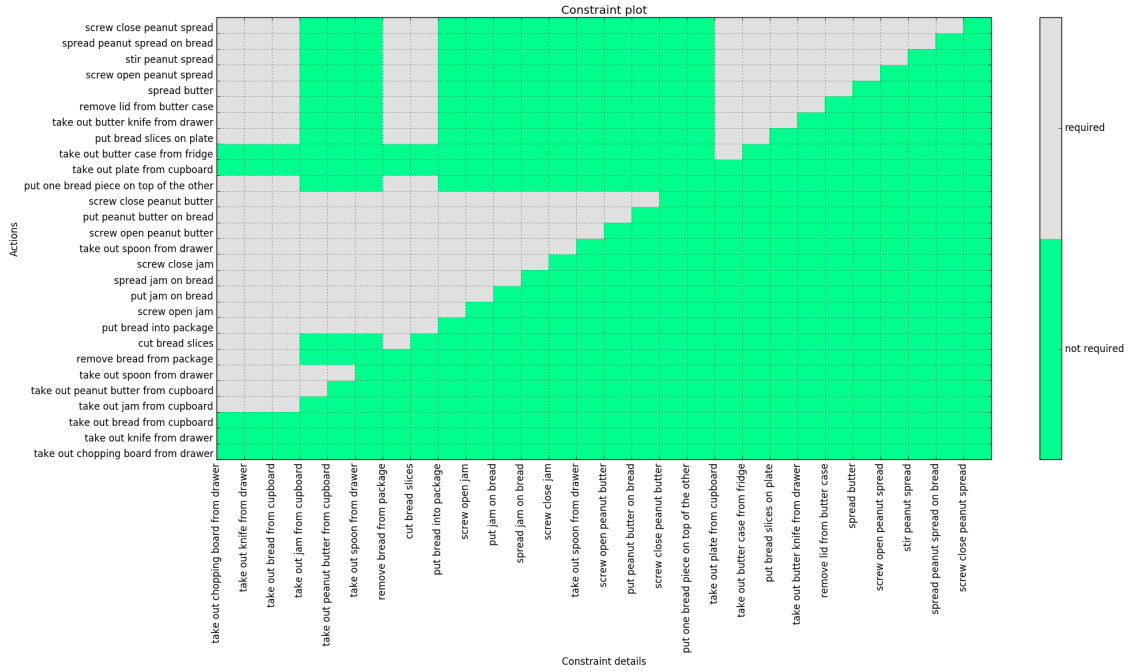

Figure 4.13: Constraint graph for use case 3 after second demonstration

put_on(cheese, bread), put(one bread piece, one bread piece).

New actions: *take_out(knife, drawer), remove(cheese, package), cut_slices(cheese), pack(cheese), take_out(tomato, fridge), wash(tomato), take_out(knife, drawer), cut_apart (tomato apart), cut_slices(tomato), put_on(butter, bread), cut_stripes(cheese), put_on(cheese, bread).*

Inference of the results

There a quite a few actions such as *take out jam from cupboard, take out peanut butter from cupboard, screw open jam, put jam on bread, spread jam on bread, screw close jam, screw open peanut butter, put peanut butter on bread, screw close peanut butter* which are present in the first demonstration but not in the second. We can see in fig 4.13 that the temporal constraints for these actions have not been modified after the second demonstration. In the third demonstration, for the new actions present all the actions preceding it have been added as the temporal constraints.

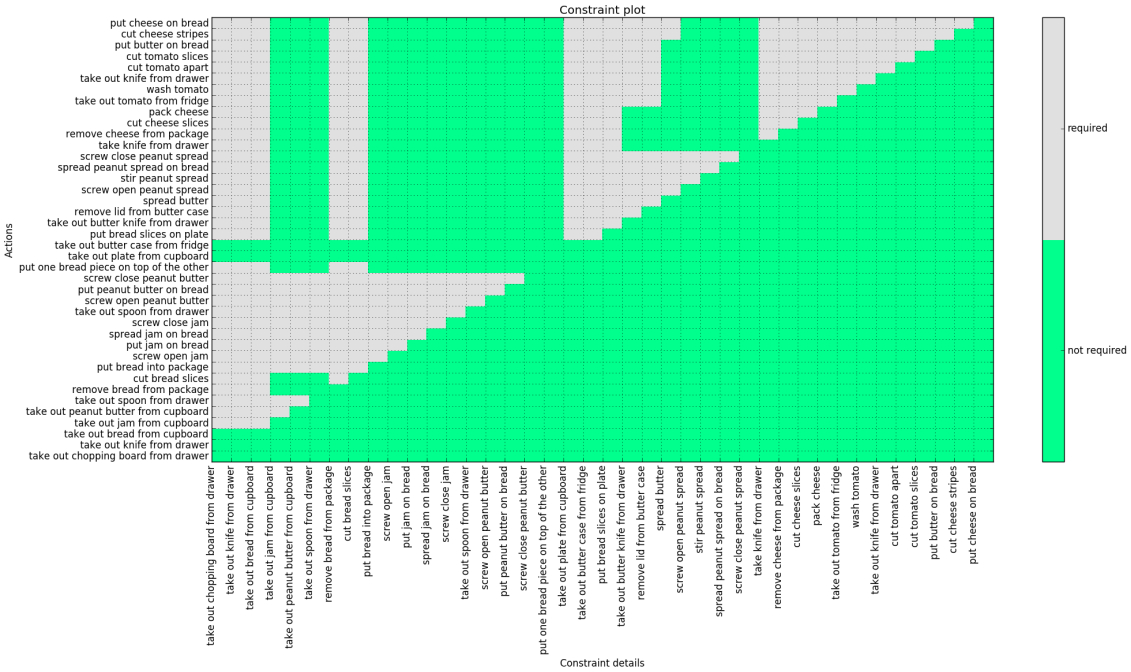

Figure 4.14: Constraint graph for use case 3 after third demonstration

4.2.4 Inference of the results for multi-step action model

From the experiments performed, it was seen that the developed algorithm was able to learn the temporal constraints for a given task for all the scenarios described in chapter 3 even when the demonstrations differed significantly.

5

Conclusions

This work has attempted to tackle the challenges such as task segmentation and generalization in learning from demonstration approach to teach robots to perform tasks by using 3D-CNN, a state of the art deep learning approach, for task segmentation and temporal constraint learning approach for task generalization.

The 3D-CNN models were evaluated on UCF-101 and MP-II cooking action dataset. It was seen that the learning rate of $3e^{-5}$ with dropout of *0.2* with batch normalization performed best on UCF-101 dataset. We obtained a test accuracy of 48.19 % with the above mentioned setting. The results of the experiments performed with dropout and batch normalization reinforce the need for using regularization techniques in 3D-CNNs. However, the 3D-CNN models were not able to learn the cooking actions when training the model from scratch on MP-II cooking action dataset. This is mainly due to lack of benchmark dataset for cooking. The datasets which are currently available are heavily biased towards certain actions and have fewer samples per class. We also noticed that when performing transfer learning by fine tuning the weights of UCF-101 to MP-II cooking action dataset the network was able to learn better even though the domains differed significantly. These results reinforce the need of stable weight initialization methods for 3D-CNNs. It also depicts the importance of using transfer learning approaches.

The action representation model developed is capable of learning the temporal constraints of a given task and improving the model given more demonstrations. We validated the model in three different use cases in increasing order of complexity and

saw that the model was able to learn the temporal constraints from the demonstrations and was capable of generalizing well for all the demonstrations.

5.1 Contributions

- We performed an analysis of the performance of 3D-CNN models on MP-II cooking activities dataset and UCF-101 dataset by varying the learning rate, dropout probabilities and using regularization techniques such as batch normalization. We also evaluated the performance of 3D-CNN on MP-II cooking action dataset by fine tuning the weights of UCF-101 dataset to MP-II cooking activities dataset.

- Inspired by the work of Ekval et al. in [10], we developed an algorithm to learn the temporal constraint present for a task given various demonstration. This algorithm can be used in a online fashion as well. We have validated the developed algorithm on 3 use cases: coffee making, mashed potato preparation and sandwich preparation.

5.2 Lessons learned

- Through the various experiments performed on the 3D-CNN models we learned the difficulty in setting the various hyper parameters such as learning rate, batch size and dropout probabilities. We also realized the importance of using regularization techniques such as dropout and batch normalization when training a deep learning network.

- When working on the algorithm for learning the temporal constraints, we also learned the various deterministic models which are used in the field of planning.

5.3 Future work

It was seen that the 3D-CNN models were overfitting to the UCF-101 dataset. Global average pooling layers have been shown to be less prone to overfitting when used instead of fully connected layers for image classification [23]. Experiments can be performed to evaluate their effectiveness in 3D-CNNs.

The action recognition module can be extended to output the parameterized action with the objects on which the action is being performed. This is also known as

video captioning. The goal in video captioning is to get a natural language sentence for a given video. This can be done by combining the output of the 3D-CNN model with a vanilla recurrent neural or LSTM network i.e the video level representation produced by the 3D-CNN can be used to train a vanilla RNN or an LSTM to predict a sentence in natural language.

Currently, we have used a temporal constraint learning approach to determine the high level representation of the given task. Here we have ignored the duration of each action. A Semi Markovian Decision Process(SMDP) can be used to model the task. Hence, taking into consideration the time taken for every action to complete. Using a probabilistic model will also help to deal with faults.

A

Action recognition datasets

A.1 Actions in UCF101 dataset

Baseball Pitch	Kayaking	Salsa Spins
Basketball Shooting	Lunges	Skate Boarding
Bench Press, Biking	Military Parade	Skiing
Billiards Shot	Mixing Batter	Skijet
Breaststroke	Nun chucks	Soccer
Clean and Jerk	Pizza Tossing	Juggling
Diving	Playing Guitar	Swing
Drumming	Playing Piano	TaiChi
Fencing	Playing Tabla	Tennis Swing
Golf Swing	Playing Violin	Throw Discus
High Jump	Pole Vault	Trampoline Jumping
Horse Race	Pommel Horse	Volleyball Spiking
Horse Riding	Pull Ups	Walking with a dog
Hula Hoop	Punch	Yo Yo
Javelin Throw	Push Ups	Apply Eye Makeup
Juggling Balls	Rock Climbing Indoor	Apply Lipstick
Jumping Jack	Rope Climbing	Archery
Jump Rope	Rowing	Baby Crawling

Balance Beam
Band Marching
Basketball Dunk
Blow Drying Hair
Blowing Candles
Body Weight Squats
Bowling
Boxing-Punching Bag
Boxing-Speed Bag
Brushing Teeth
Cliff Diving
Cricket Bowling
Cricket Shot
Cutting In Kitchen
Field Hockey Penalty
Floor Gymnastics
Frisbee Catch
Front Crawl
Hair cut
Hammering
Hammer Throw
Handstand Pushups
Handstand Walking
Head Massage
Ice Dancing
Knitting
Long Jump
Mopping Floor
Parallel Bars
Playing Cello
Playing Daf
Playing Dhol
Playing Flute
Playing Sitar
Rafting
Shaving Beard
Shot put
Sky Diving
Soccer Penalty
Still Rings
Sumo Wrestling
Surfing
Table Tennis Shot
Typing
Uneven Bars
Wall Pushups
Writing On Board

A.2 MP-II cooking activities dataset

Background activity
change temperature
cut apart
cut dice
cut in
cut off ends
cut out inside
cut slices
cut stripes
dry
fill water from tap
grate
lid: put on
lid: remove
mix
move from X to Y
open egg
open tin
open/close cupboard
open/close drawer
open/close fridge
open/close oven
package X
peel
plug in/out
pour
pull out
puree
put in bowl
put in pan/pot
put on bread/dough
put on cutting-board
put on plate
read
remove from package
rip open
scratch off
screw close
screw open
shake
smell
spice

spread
squeeze
stamp
stir
strew
take & put in cupboard
take & put in drawer
take & put in fridge
take & put in oven
take & put in spice holder
take ingredient apart
take out from cupboard
take out from drawer
take out from fridge
take out from oven
take out from spice holder
taste
throw in garbage
unroll dough
wash hands
wash objects
whisk
wipe clean

References

[1] Keras documentation[online], Accessed Jan 04, 2019. URL `https://keras.io/getting-started/sequential-model-guide/`.

[2] 3d pooling [online], Accessed Jan 04, 2019. URL `http://cs231n.github.io/convolutional-networks/`.

[3] Convolution operation with sobel filter [online], Accessed Jan 13, 2019. URL `https://medium.freecodecamp.org/an-intuitive-guide-to-convolutional-neural-networks-260c2de0a050`.

[4] N. Abdo, H. Kretzschmar, L. Spinello, and C. Stachniss. Learning Manipulation Actions from a Few Demonstrations. *IEEE International Conference on Robotics and Automation, Karlsruhe*, pages pp. 1268–1275., 2013.

[5] S.R. Ahmadzadeh, A. Paikan, F. Mastrogiovanni, L. Natale, P. Kormushev, and D.G. Caldwell. Learning symbolic representations of actions from human demonstrations. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3801–3808. IEEE, 2015.

[6] A. Attia and S. Dayan. Global overview of imitation learning. *arXiv preprint arXiv:1801.06503*, 2018.

[7] Joao Carreira and Andrew Zisserman. Quo vadis, action recognition? a new model and the kinetics dataset. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 4724–4733. IEEE, 2017.

[8] J.M. Chaquet, E.J. Carmona, and A. Fernández-Caballero. A survey of video datasets for human action and activity recognition. *Computer Vision and Image Understanding*, 117(6):633–659, 2013.

[9] A. Diba, M. Fayyaz, V. Sharma, A. H. Karami, M. M. Arzani, R. Yousefzadeh, and L. Van Gool. Temporal 3d convnets: New architecture and transfer learning for video classification. *arXiv preprint arXiv:1711.08200*, 2017.

[10] S. Ekvall and D. Kragic. Learning Task Models from Multiple Human Demonstrations. *International Symposium on Robot and Human Interactive Communication*, pages 358–363, 2006.

[11] C. Feichtenhofer, A. Pinz, and A. 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.

[12] X. Glorot, A. Bordes, and Y. Bengio. Deep sparse rectifier neural networks. In *Proceedings of the fourteenth international conference on artificial intelligence and statistics*, pages 315–323, 2011.

[13] B. Hayes and B. Scassellati. Discovering task constraints through observation and active learning. In *IEEE International Conference on Intelligent Robots and Systems*, pages 4442–4449, 2014.

[14] B. Hayes and B. Scassellati. Autonomously constructing hierarchical task networks for planning and human-robot collaboration. In *Proceedings - IEEE International Conference on Robotics and Automation*, volume 2016-June, pages 5469–5476. IEEE, 2016.

[15] G.E. Hinton, N. Srivastava, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*, 2012.

[16] G. Huang, Z. Liu, L. Van Der Maaten, and K. Q. Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

[17] S. Ji, W. Xu, M. Yang, and K. Yu. 3D Convolutional Neural Networks for Human Action Recognition. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 35(1):221–231, 2013.

[18] A. Karpathy, G. Toderici, S. Shetty, T. Leung, R. Sukthankar, and F. Li. Large-scale video classification with convolutional neural networks. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1725–1732, 2014.

[19] S. Kim and R. Casper. Applications of convolution in image processing with matlab. *University of Washington*, 2013.

[20] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proceedings of the 25th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'12, pages 1097–1105, USA, 2012. Curran Associates Inc.

[21] A. Krizhevsky, I. Sutskever, and G.E. Hinton. Imagenet classification with deep convolutional neural networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1097–1105. Curran Associates, Inc., 2012.

[22] H. Kuehne, H. Jhuang, E. Garrote, T. Poggio, and T. Serre. Hmdb: a large video database for human motion recognition. In *Computer Vision (ICCV), 2011 IEEE International Conference on*, pages 2556–2563. IEEE, 2011.

[23] M. Lin, Q. Chen, and S. Yan. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

[24] Y. Liu, A. Gupta, P. Abbeel, and S. Levine. Imitation from observation: Learning to imitate behaviors from raw video via context translation. In *2018 IEEE International Conference on Robotics and Automation (ICRA)*, pages 1118–1125. IEEE, 2018.

[25] K. Mourão, L. Zettlemoyer, R.P Petrick, and M. Steedman. Learning strips operators from noisy and incomplete observations. *arXiv preprint arXiv:1210.4889*, 2012.

[26] M. N. Nicolescu and M.J. Mataric. Natural methods for robot task learning. *Proceedings of the second international joint conference on Autonomous agents and multiagent systems - AAMAS '03*, page 241, 2003.

[27] M.N. Nicolescu and M.J. Mataric. Learning and interacting in human-robot domains. *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, 31(5):419–430, 2001.

[28] M.N. Nicolescu and M.J. Matarić. A hierarchical architecture for behavior-based robots. In *Proceedings of the first international joint conference on Autonomous agents and multiagent systems part 1 - AAMAS '02*, page 227, 2002.

[29] Diederik P.K. and Jimmy B. Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*, 2014.

[30] A. S. Razavian, H. Azizpour, J. Sullivan, and S. Carlsson. Cnn features off-the-shelf: an astounding baseline for recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, pages 806–813, 2014.

[31] M. Rohrbach, S. Amin, M. Andriluka, and B. Schiele. A database for fine grained activity detection of cooking activities. In *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, pages 1194–1201, 2012.

[32] S. Ruder. An overview of gradient descent optimization algorithms. *arXiv preprint arXiv:1609.04747*, 2016.

[33] C. Sammut. *Behavioral Cloning*, pages 93–97. Springer US, Boston, MA, 2010.

[34] A. Shimada, D. Kondo, K.and Deguchi, G. Morin, and H. Stern. Kitchen Scene Context Based Gesture Recognition: A Contest in ICPR2012. pages 168–185. Springer, Berlin, Heidelberg, 2013.

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

[36] K. Soomro, A.R. Zamir, and M. Shah. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402*, 2012.

[37] R.S. Sutton and A.G. Barto. Reinforcement learning: An introduction. 2011.

[38] M. Tenorth, J. Bandouch, and M. Beetz. The TUM Kitchen Data Set of everyday manipulation activities for motion tracking and action recognition. In *2009 IEEE 12th International Conference on Computer Vision Workshops, ICCV Workshops*, pages 1089–1096. IEEE, 2009.

[39] F. Torabi, G. Warnell, and P. Stone. Behavioral cloning from observation. In *IJCAI*, 2018.

[40] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri. Learning spatiotemporal features with 3D convolutional networks. In *Proceedings of the IEEE International Conference on Computer Vision*, volume 2015 Inter, pages 4489–4497, 2015.

[41] L. Wang, Y. Xiong, Z. Wang, Y. Qiao, D. Lin, X. Tang, and L. Van Gool. Temporal segment networks: Towards good practices for deep action recognition. In *European Conference on Computer Vision*, pages 20–36. Springer, 2016.

[42] X. Wang. *Learning planning operators by observation and practice*. PhD thesis, Carnegie Mellon University, 1996.

[43] Q. Yang, K. Wu, and Y. Jiang. Learning action models from plan examples using weighted max-sat. *Artificial Intelligence*, 171(2-3):107–143, 2007.

[44] Y. Yang, Y. Li, C. Fermüller, and Y. Aloimonos. Robot Learning Manipulation Action Plans by "Watching" Unconstrained Videos from the World Wide Web. *Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*, pages 3686–3692, 2015.

[45] Y. Zhu, Z. Lan, S. Newsam, and A.G. Hauptmann. Hidden two-stream convolutional networks for action recognition. *arXiv preprint arXiv:1704.00389*, 2017.

[46] H.H. Zhuo and S. Kambhampati. Action-model acquisition from noisy plan traces. In *IJCAI*, pages 2444–2450, 2013.