Recognizing Compositional Actions in Videos with Temporal Ordering

by

Vivek Kumar Maskara

A Thesis Presented in Partial Fulfillment
of the Requirements for the Degree
Master of Science

Approved December 2021 by the
Graduate Supervisory Committee:

Hemanth Venkateswara, Co-Chair
Troy McDaniel, Co-Chair
Hasan Davulcu

ARIZONA STATE UNIVERSITY

May 2022

ABSTRACT

In some scenarios, true temporal ordering is required to identify the actions occurring in a video. Recently a new synthetic dataset named CATER, was introduced containing 3D objects like sphere, cone, cylinder etc. which undergo simple movements such as slide, pick & place etc. The task defined in the dataset is to identify compositional actions with temporal ordering. In this thesis, a rule-based system and a window-based technique are proposed to identify individual actions (atomic) and multiple actions with temporal ordering (composite) on the CATER dataset. The rule-based system proposed here is a heuristic algorithm that evaluates the magnitude and direction of object movement across frames to determine the atomic action temporal windows and uses these windows to predict the composite actions in the videos. The performance of the rule-based system is validated using the frame-level object coordinates provided in the dataset and it outperforms the performance of the baseline models on the CATER dataset. A window-based training technique is proposed for identifying composite actions in the videos. A pre-trained deep neural network (I3D model) is used as a base network for action recognition. During inference, non-overlapping windows are passed through the I3D network to obtain the atomic action predictions and the predictions are passed through a rule-based system to determine the composite actions. The approach outperforms the state-of-the-art composite action recognition models by 13.37% (mAP 66.47% vs. mAP 53.1%).

ACKNOWLEDGMENTS

I would like to express my gratitude to Dr. Hemanth Venkateswara for his constant support and guidance throughout the project. His vision and ideas constantly motivated me to push the boundaries, and I am grateful for the opportunity to work with him.

I want to thank Dr. Troy McDaniel and Dr. Hasan Davulcu, for their continuous guidance and support during the research work and the thesis.

I want to thank Dr. Mark Naufel for supporting me throughout my Masters degree. He has been a great mentor and he inspires me everyday. I also want to thank all my colleagues at the Luminosity Lab. Working with everyone in the lab has been a tremendous learning experience for me.

I would also like to thank my parents, Santosh Kumar Maskara and Suman Maskara, and my fiancée Utkarsha Bakshi for everything.

TABLE OF CONTENTS

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Chapter 1

INTRODUCTION

Deep learning based architectures have made immense progress in recent years and have become quite accurate at tasks involving image classification and object detection. The results obtained from these tasks have motivated the research community to attempt more complex problems such as action recognition. Action recognition in videos is a standard Computer Vision problem and has been well studied. In videos, in addition to the individual frames, the temporal component provides important clues for identifying the action reliably. Moreover, real world videos offer multiple additional challenges including jittering, camera motion and varying scene background. These problems are unique to the videos and require additional effort for tackling them efficiently.

In video action recognition, the fundamental goal is to analyze a video to identify the actions taking place in the video. Essentially a video has a spatial aspect to it i.e. the individual frames and a temporal aspect i.e. the ordering of the frames. Some actions (eg. standing, running, etc.) can potentially be identified by using just a single frame but for more complex actions(eg. walking vs running, bending vs falling) might require more than 1 frame's information to identify it correctly. Local temporal information plays an important role in differentiating between such actions. Moreover, for some use cases, local temporal information isn't sufficient and you might need long duration temporal information to correctly identify the action or classify the video. Identifying such actions in videos find applications in several scenarios including video surveillance, person tracking and providing response in emergency situations. In a real world scenario, videos contain frequent long term occlusion and

contain multiple actors performing different actions simultaneously. It is important not only to identify these actions but also to understand the order of these actions. Moreover, these systems can also find usage in live game referral systems where it is critical to identify the order of actions. For eg., the user might be interested in knowing whether a first a goal was scored or was a foul made first. During person tracking, frequent occlusion is common if the actor is in a busy surrounding and identifying his actions would require understanding the long term occlusion well.

In this thesis, I have taken the first step for identifying actions with temporal ordering. The goal was to conduct the experiments in a controlled environment so that the results can be analyzed carefully and conclusions can be drawn from it. For this thesis, all the experiments were performed using a synthetic dataset containing around 5500 videos. All the videos in the dataset consisted of simple objects such as a cone, cylinder, sphere or a cube. The presence of simple 3D objects in the scene simplifies the task of identification and re-identification in the videos. In real world datasets, if the objects have a certain amount of variations, it becomes difficult for the network to learn all those subtle variations. For eg., a network would require thousands of images of a cat or a dog to become precise in the task of identifying a cat from a dog. Moreover, all the videos consisted of the same background so that scene bias could be eliminated. In real world datasets, often the background of the scene offers important clues on the action taking place. For eg. if the network sees a football field, it can easily infer that the actor is playing football or else a swimming pool can indicate the act of swimming. The synthetic dataset with a constant background eliminates these issues and the network can entirely focus on the objects and the actions being performed by those objects.

The CATER (Girdhar and Ramanan (2019)) dataset contains synthetically generated videos containing simple 3D objects of different shapes and sizes placed before

a solid background. The objects in the scene perform a subset of 4 atomic actions: pick place, slide, rotate and contain. The dataset contains a total of 14 unique atomic actions and identifying these actions constitute Task 1. Since multiple of these actions can take place in a video, identifying the temporal order between the actions constitutes Task 2. The ordering is specified using three temporal intervals i.e. before, during and after. Using these three temporal intervals and combining it two of the 14 atomic actions, a total of 301 unique composite actions are obtained. Task 2 is setup as a multi-label classification problem for identifying which of these composite actions occur in the video. Moreover, the dataset also defines a special object i.e. the snitch and identifying its final location in the scene constitutes Task 3. In this thesis, Task 1 and Task 2 was attempted. Chapter 3 provides more details about the dataset and the tasks that were attempted. Moreover, for both Task 1 and Task 2, experiments were performed with and without camera motion.

The actions defined in the dataset have a distinct spatiotemporal trajectory which can be used to distinguish between different classes of actions. If the actions have a unique trajectory then the problem can be reduced to estimating the location of the objects in each of the frames and then using the (x, y) coordinates to classify the trajectory. To validate our assumption, a handcrafted rule based system was developed that can be fed with the (x, y) coordinates of each of the objects and can output the action predictions. The algorithm starts by finding all windows where a particular object's location changes and then these windows are categorized into different actions based on the type of motion that was observed. The algorithm then eliminates any overlapping windows based on a priority function. Finally, scene level action predictions are made and the results are compared with the ground truth information. Similarly for composite actions, the action windows are used to define a temporal relationship between a pair of actions.

The system was tested using the frame level annotations from the training data and it was found that the system surpasses the state of the art performance for both atomic and composite actions. It provided the confidence to pursue this method further and derive the (x, y) coordinates from the videos using object tracking. Since the dataset contains simple shapes without a lot of inter-class variations, a standard object detection network was used for getting the bounding boxes. These bounding boxes were then input to a object tracking module to get the trajectories for the objects in the scene. Chapter 4 explains the process in detail and Chapter 5 describes the results from the experiment.

In this thesis, experiments using other deep learning models such as I3D and Non-local neural nets were performed for atomic-action recognition, as described in 4. Moreover, multiple experiments were performed using transformer based networks for atomic action recognition. The transformer based networks failed to effective learn the atomic actions in the dataset and its inability to learn can be accounted to the lack of a large dataset available for training.

The experiments using I3D networks for atomic action recognition closely matched the state of art performance as described in the original paper for CATER (Girdhar and Ramanan (2019)) dataset. It provided the motivation to come up with a window based technique where the whole video is divided into multiple clips and the training is performed on these clips for atomic action recognition. The predictions from the model is then passed through a simple rule based system to identify the temporal ordering of the actions in the video. Multiple experiments were performed using these approach including different window sizes, random windows and moving windows. This method surpassed the performance for composite action recognition i.e. Task 2 as described in the CATER dataset.

The contribution of this work is three-fold. First, the thesis proposes, algorithms

for a rule-based for identifying both atomic and composite actions. The algorithms validate the hypothesis that the object tracks obtained by a robust tracking algorithm will perform exceedingly well on this dataset. The algorithms when fed with frame wise object coordinates from the training data, outperform the state of the art results as reported in the CATER (Girdhar and Ramanan (2019)) dataset. If the same rule based algorithms are fed with tracks from a robust multi-object tracker, it would perform equally well.

Second, the thesis provides a script for generating labels for atomic actions for any non-overlapping windows or even random sequences of varying sizes from the videos. This script could aid in future research by making ground truth labels easily available for any desired sequence in the videos. The windows in the videos refer to a set of consecutive non-overlapping clips with each clip having multi actions taking place. Similarly, a random sequence refers to a sequence of frames in a video of the specified length. The script for random sequence also allows the user to specify the number of samples that needs to be picked from each video.

Third, the thesis proposes a new approach for identifying composite actions in videos. It proposes an approach where first the whole video is used for training to learn atomic actions in the video. Next, the video is split into multiple non-overlapping windows and the network is trained on these clips. The labels for these clips are generated using the above script using the frame level scene annotations available with the original dataset. Once the model outputs the atomic actions for each of these clips, a simple rule based system is used for determining the composite actions in the scene. This approach surpasses the existing state of the art performance for Task 2 by an impressive margin. It must be noted that this technique aims to solve for this specific dataset and might not be efficient for other datasets. Moreover, for real world datasets, it might be difficult to identify exact ground truth labels for

the different sub-clips from the video.

Chapter 2

RELATED WORK

The problem of action recognition in videos can vary widely and there's no single approach that suits all the problem statements. Traditional approaches to action recognition rely on object detection, pose detection, dense trajectories, or structural information. More recent approaches have relied on 3D-CNN and transformer based approaches. Video action recognition is a widely researched field and numerous works have been published in the last few years exploring different approaches under a variety of settings and applicable to different datasets.

2.1 Trajectory Based Methods

Convolutional Neural Networks(CNN) extracts the features from each frame and pool the features from multiple frames to get a video-level prediction. The drawback of this approach is that it fails to capture sufficient motion information. Motion information can be captured by combining optical flow containing short-term motion. In addition to RGB and optical flow, information from other modalities such as audio, pose, and trajectory can also be used.

Dense Trajectories(Wang *et al.* (2011)) concatenated trajectory descriptors with appearance features from each frame and tracks them based on displacement information. Their approach combines dense sampling with feature tracking and introduces an efficient solution for removing camera motion. They compute the motion boundary descriptors along the dense trajectories which help in tackling the camera motion. The motion information in the dense trajectories is leverages by calculating the features within a space-time volume around the trajectory. Another approach

using trajectory information by Wang and Schmid (2013), explicitly estimates the camera motion to improve the dense trajectories. For estimating the camera motion, they match feature points between frames using SURF descriptors and dense optical flow. It also uses state of the art human detectors to remove potentially inconsistent matches during camera motion estimation.

Two stream CNN(Simonyan and Zisserman (2014)) uses separate spatial and temporal recognition streams based on ConvNets. The two-stream network consists of two separate subnetworks, where one is for raw images and the other is for stacked optical flow, respectively, and captures spatiotemporal information by fusing the softmax scores of two streams. The temporal stream uses optical flow displacement fields for capturing the trajectory information.

A spatio-temporal representation can be constructed by fusion motion and appearance information in the way of two streams. It would work well for short duration clips, but would not be able to capture long-term temporal dynamics.

2.2 Spatio-temporal Networks

3D Convolutional Neural Networks for Human Action Recognition (Ji *et al.* (2013)) proposes to perform 3D convolutions to extract spatial and temporal features from the video. It proposes the use of a 3D-CNN model for this purpose which generates multiple channels of information from adjacent video frames, performing convolution and subsampling separately in each channel. It also proposes the uses of auxiliary outputs for regularizing the 3D CNN models.

Another work, ConvNet Architecture Search for Spatiotemporal Feature Learning (Tran *et al.* (2017)) uses 3D CNNs as feature extractors. The convolution is applied on a spatio-temporal cube ie. it includes multiple frames. They use a SVM classifier (Hearst *et al.* (1998)) on top of the feature extractor as a classifier. The network

performed decently for short videos but long-range temporal modeling was still not performing well with this network.

Large-scale Video Classification with Convolutional Neural Networks (Karpathy *et al.* (2014)), introduces early, late and slow fusion for fusing time information in CNN models. Their experiments indicate that slow fusion performs better than other methods. It also introduces a multi-resolution architecture to reduce the computation cost. This architecture uses 2 separate streams which processes the image at 2 different spatial resolutions. The issue with their approach was that very little performance improvement was achieved using their approach.

Recurrent neural networks (RNNs), especially long short-term memory (LSTM), achieved impressive results in the sequence tasks due to the ability of long-term temporal modeling, so an alternative strategy is to adopt LSTM to model dynamics of frame-level features. However, most existing LSTM-based approaches do not make the distinction between various parts of video frames. LRCN (Donahue *et al.* (2015)) is a recurrent convolutional architecture, which cascaded a CNN with a recurrent model into a unified model. CNN was used to extract features of each frame, and then, these features were fed into LSTM step by step for modeling dynamics of the feature sequence so that it could learn video level representation in both spatial and temporal dimensions. Beyond short snippets(Ng *et al.* (2015)), combined the temporal feature pooling architecture with LSTM to allow the model to accept arbitrary-length frames. Beyond frame level CNN (Wang *et al.* (2017)) utilized a deep 3-D-CNN to process salient-aware clips and fed the features extracted from the fully connected layer of a 3-D-CNN into LSTM for action recognition. According to the spatial–optical data organization(Yuan *et al.* (2018)), synthesized motion trajectories, optical, and video segmentation into spatial–optical data and used a two-stream 3-D CNN to process synthetic data and RGB data separately.

2.3 Transformer Based Networks

Attention is all you need (Vaswani *et al.* (2017)) proposed a network based on purely attention mechanisms for language translation task. Gradually, more work was done for finding applications for transformer based models for vision tasks. One of the first works proposing the use of transformers for image classification was the ViT network (Dosovitskiy *et al.* (2021)). It proposes dividing the frame into small patches and then passing these patches through a linear layer to get embeddings. These embeddings are then stacked together and passed through a standard transformer network to learn a class token embedding. These embeddings are passed through a MLP layer for image classification.

The ViViT paper (Arnab *et al.* (2021)) extended the ViT approach to videos. It proposed multiple techniques for the temporal extension. One of the approaches proposed in the ViViT paper was to first use a standard ViT network on the all frames of the video and then combining the class token embedding outputs. The combined class token embedding outputs are passed through a temporal transformer and then a MLP layer to learn the actions occurring in the video. Fig. 4.7 shows the network architecture for this approach.

More recent architectures have focused on using attention mechanisms for picking salient parts of the video. This helps in overcoming the limitation of LSTMs which didn't distinguish between various parts of the video. 3D CNN RNN encoder decoder model(Yao *et al.* (2015)) uses attention mechanism for effective video description as it allows the usage features obtained using global analysis of static frames. Visual attention(Sharma *et al.* (2016)) proposes a soft attention based model for action recognition. The model learns to focus selectively on the important parts of the video. Global and Local Knowledge-Aware Attention Network(Zheng *et al.* (2021))

incorporates two types of attention mechanisms called statistic-based attention (SA) and learning-based attention (LA) to attach higher importance to the crucial elements in each video frame. Attention Clusters(Long *et al.* (2017)) uses multiple attention mechanisms units(called attention clusters) to capture information from multiple modalities. Video Action Transformer Network(Girdhar *et al.* (2019)) introduces a transformer based architecture that learns to focus on hands and faces which is often crucial in differentiating between actions. They use an action transformer as input for the video feature representation and the box proposal from RPN and maps it into query and memory features.

Chapter 3

DATASET

The experiments in this thesis are conducted on the CATER (Girdhar and Ramanan (2019)) dataset. CATER is an extension of CLEVR (Johnson *et al.* (2016))

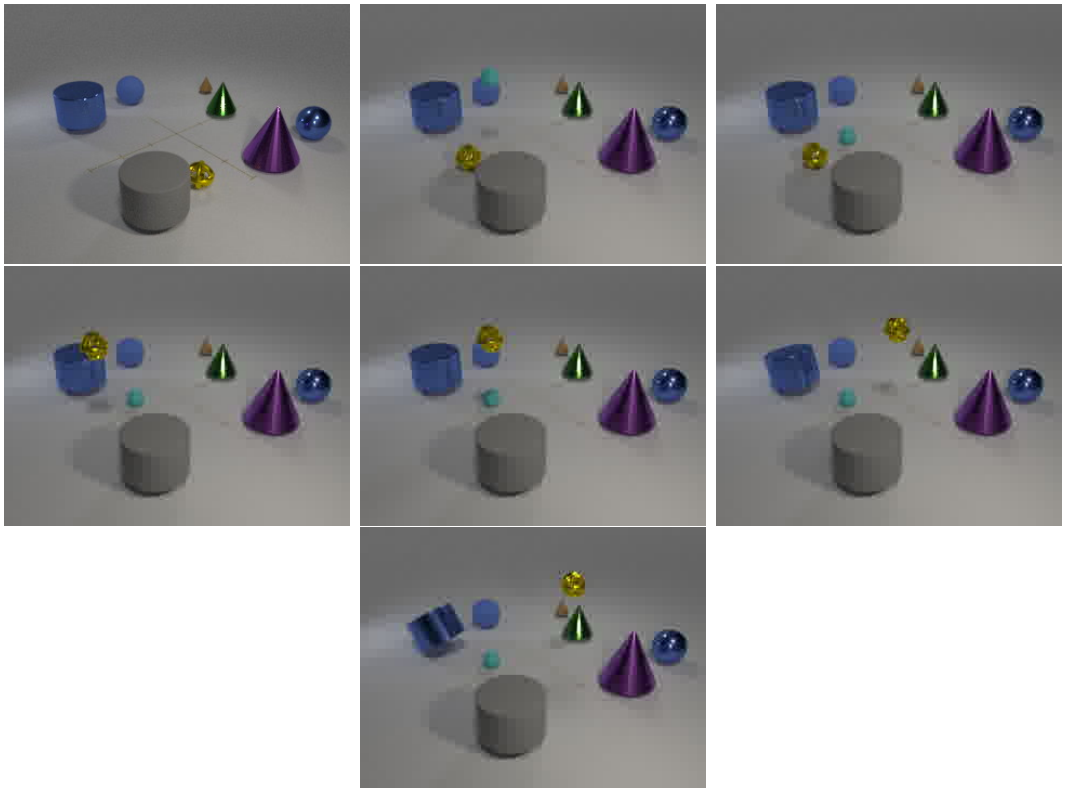


Figure 3.1: Sample Frames from a Video in the Cater Dataset showing the movements of different 3D objects (Source: CATER Dataset (Girdhar and Ramanan (2019))). In this sample, multiple objects are performing different actions simultaneously. It can be noticed that the blue cylinder is rotating along its horizontal axis from the top-left frame to the bottom-right frame. Moreover, the golden snitch can been being picked from its place and placed to another location in the scene.

and it can be used for understanding compositional and temporal actions in videos. It is a synthetic dataset and it contains simple 3D objects such as a sphere, cone, cylinder and a snitch. Each of these objects can afford one or multiple actions such as slide, pick-place, contain and rotate. mAP is used as the performance metric for both tasks. Fig. 3.1 shows some sample frames taken from a training video.

The dataset contains 5500 videos which are split into training and validation set(80:20). Each video is a 300-frame 320x240px video at 24 FPS. During data generation, each video is divided into 30 frame slots and each action is contained within these slots. Actions are iteratively added to at max K objects in these slots in an random order. For Task 1 and Task 2, K=2 is used to avoid too many actions happening at the same time. For Task 3, K=N is used since the goal is to only find the final location of the special object.

The original paper defines 3 different tasks of varying difficulties for action classification and localization.

Task 1: Atomic action recognition This task is to produce 14 different probability values to indicate the likelihood of that action taking place. A single video can have multiple actions taking place, so Task 1 is designed as a multi-label classification problem. The performance for this task is evaluated using mean average precision(mAP), computed by taking a mean of average precision(AP) over all the classes. This is a popular evaluation metric for multi-label action classification datasets (Sigurdsson *et al.* (2016), Gu *et al.* (2018)).

Task 2: Composite action recognition This task is to predict the occurrence of all the composite actions occurring in the video out of the 301 possible actions. It takes pairs of 2 out of 14 atomic actions and joins it using a temporal relation(i.e. before, during, after) to obtain all the different action classes. Similar to Task 1, this problem is also defined as a multi-label classification problem. The performance for

this task is evaluated using mAP metric similar to what was used for Task 1.

Task 3: Snitch Localization The final task defined in the dataset requires predicting the final location of the special object called the snitch. Since, the snitch can be occluded from view in the final frame, it is not a trivial task to identify its final location. The frame is quantized into 36 equally equally sized cells and the problem is defined as a classification problem where the goal is to predict the cell in which the snitch is located in the final frame.

In this thesis, the first two tasks were attempted and Chapter 4 describes the methodology used and Chapter 5 describes the results.

Chapter 4

PROPOSED METHOD

4.1 Rule Based System

Looking at the actions defined in the dataset, one can observe that most of the actions have a distinctive trajectory in space and it can be exploited to distinguish between different action classes. Fig. 4.3 shows the trajectories for different atomic actions defined in the dataset. This characteristic motivated us to track all the objects

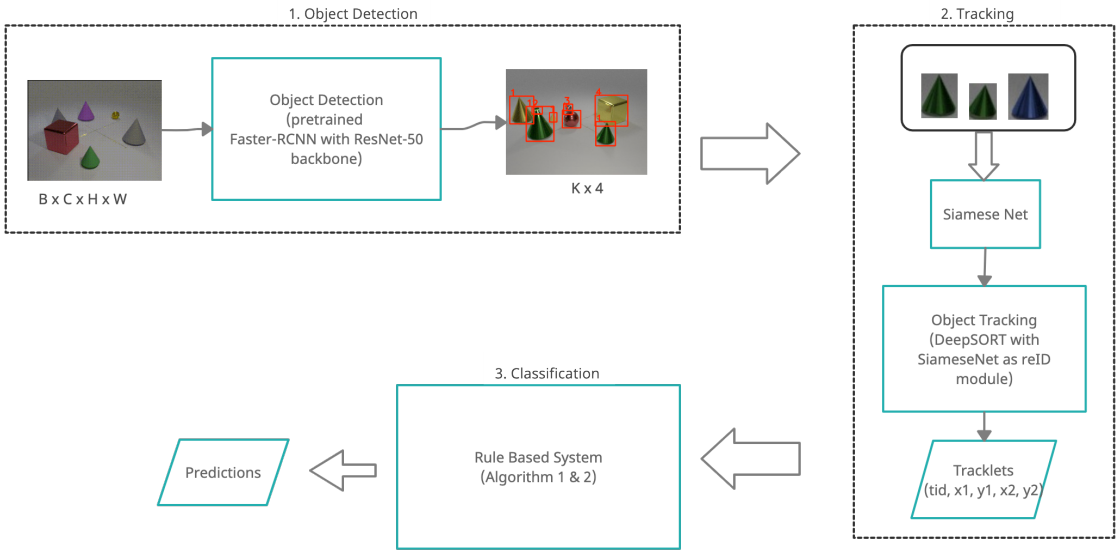


Figure 4.1: Architecture for Rule Based System: (1) Individual frames are passed through a pre-trained Faster-RCNN network with a ResNet-50 backbone to detect all the objects in the frame. (2) A multi-object tracking algorithm is used to obtain the trajectories for all the objects in the scene. (3) The trajectory information is passed through a rule-based system to determine atomic and composite actions in the video

in the scene and use handcrafted rules to classify the trajectories into different action classes. Initially, a Faster-RCNN Ren *et al.* (2016) model is trained on a small set of images to identify all the objects in the scene. The resulting bounding boxes are then passed to the DeepSORT(Wojke *et al.* (2017)) algorithm to obtain object trajectories. These object trajectories are input to a rule based system that classifies the action classes. The methodology is described in detail in the following sections.

4.1.1 Object Detection

The dataset comprises of simple 3D objects of varying colors and sizes and is set against the same background for all the videos. Since, the objects do not exhibit a lot of inter-class variation, we manually annotated 100 images extracted from different videos using(Tzutalin (2015)). These images were then used for training a standard Faster-RCNN (Ren *et al.* (2016)) model for predicting bounding boxes and class labels.

Faster-RCNN (Ren *et al.* (2016)) takes an input image and passes it through a pre-trained CNN network to obtain a convolutional feature map. This feature map is passed through a Region Proposal Network (RPN) to obtain a fixed number of regions with probability of containing objects in them. Next, Region of Interest (RoI) Pooling is applied to extract features from each of the bounding boxes given out by the RPN. Finally, a classifier is used on the extracted feature map to classify the it as an object or a background. Fig. 4.2 shows the architecture for Faster-RCNN.

We used a ResNet-50(He *et al.* (2015)) model as a feature extractor for the Faster-RCNN model. The feature map obtained from it is then passed through a Region Proposal Network(RPN) which outputs a predefined number of regions where objects might exist. RPN uses fixed sized anchor boxes placed uniformly throughout the image. Region of Interest(ROI) Pooling is applied on each of these regions to

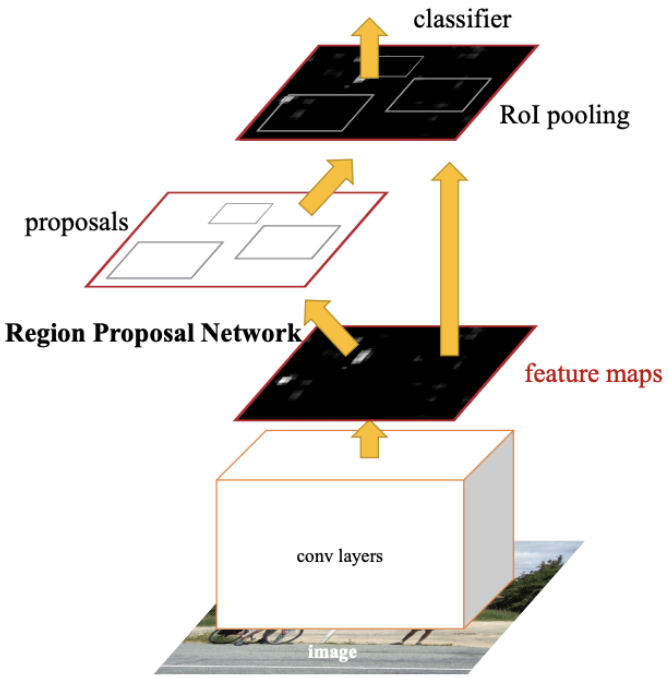


Figure 4.2: Architecture for Faster-RCNN (Source: Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks (Ren *et al.* (2016))). The figure shows that the image is passed through the conv layers which give out a feature map. The feature map is passed through a Region Proposal Network (RPN) and a Region of Interest (ROI) pooling layer to extract the feature map for each proposal.

extract features. Finally, the R-CNN module either classifies it as an object or as a background.

We perform multiple experiments by tweaking the input channels, performing data augmentation and using pre-trained weights etc. to iteratively increase the accuracy of the model.

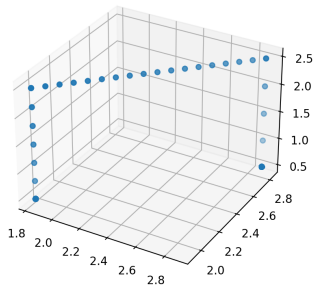
4.1.2 Object Re-identification

Efficient tracking of multiple objects in the scene requires accurate re-identification of the object across frames. We will be training a Siamese Net(Koch *et al.* (2015)) with a triplet loss function to group together similar objects and simultaneously learn to keep the representation of other objects away from the first group. During training a set of 3 images are picked, where the first image is one that is being trained, the second image belongs to the same class as the first image(positive class) and the third image belongs to the negative class i.e. it can be from any class apart from the first image's class. The loss function of the convolutional neural network tries to minimize the cosine distance between the anchor and the positive class image and maximises the cosine distance between the anchor and the negative class images.

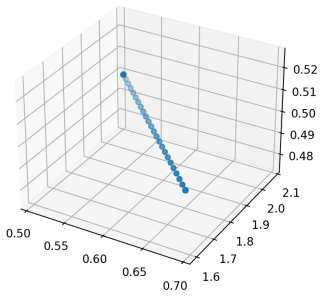
Since, object detection gave a very high accuracy, the training images were obtained by running the detection model on frames of the videos. The CATER dataset defines 192 distinct object classes based on varying objects, their size, color and texture. The object detection module was run until 300 instances of each object categories were obtained. We manually sifted across the resulting crops to remove all incorrectly labeled images. The model was then trained for 40 epochs using a triplet loss function using 80% images as training images.

4.1.3 Object Tracking

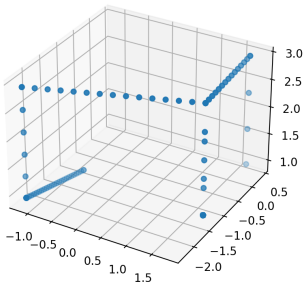
We used DeepSORT Wojke *et al.* (2017) for multi-object tracking as its a simple and elegant framework for tracking multi objects simultaneously. The model takes in only the set of bounding boxes for each of the frames. It uses the Siamese Net as a feature extractor for objects in the scene. The feature map is used to calculate the distance D between two objects as shown in Eq. 4.1. D_k represents the Mahalanobis



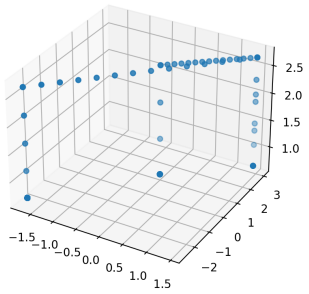
(a) Pick Place



(b) Slide



(c) Pick Place and Slide



(d) Pick Place twice

Figure 4.3: Plot Showing Object Trajectories for Atomic Actions. The plots clearly indicate each of these actions have a distinct trajectory and this information can be exploited to determine the actions using the object trajectories

distance, D_a represents the cosine distance and Λ represents the weighting factor.

The network returns tracks for each of the objects in the scene along with all the necessary information about the object's state.

$$D = \Lambda * D_k + (1 - \Lambda) * D_a \tag{4.1}$$

Algorithm 1 Get Atomic Action Windows From Scene

procedure AtomicActionWindows($sceneInfo$)
  $sceneObjects \leftarrow$ GetSceneObjects($sceneInfo$)
  $sceneActionWindows \leftarrow < k, v >$
  for all $obj_1 : sceneObjects$ **do**
    $actionWins \leftarrow < k, v >$
    $(x_1, y_1, z_1), name_1, shape_1 =$ GetObjCoords(obj_1)
    $actionWins[slide] =$ GetSlideWins($shape_1, x_1, y_1, z_1$)
    $actionWins[pickPlace] =$ GetPickPlaceWins($shape_1, x_1, y_1, z_1$)
    if $shape_1$ is cone **then**
      for all $obj_2 : sceneObjects$ **do**
        if $indexOf(obj_2) = indexOf(obj_1)$ **then** continue
        end if
        $(x_2, y_2, z_2), name_2, shape_2 =$ GetObjectCoords($objectDetails$)
        $actionWins[contain] =$ GetContainWins($shape_2, x_2, y_2, z_2$)
      end for
    end if
    $sceneActionWindows[name_1] = actionWins$
  end for
end procedure

Algorithm 2 Get Atomic Action Predictions

```
1: procedure AtomicActionPredictions(sceneActionWindows)
2:     predActions ← list()
3:     for all obj : sceneActionWindows do
4:         movements ← sceneActionWindows[obj]
5:         if contain in movements then
6:             for all containWins : movements[contain] do
7:                 Remove overlapping pickPlace and slide movements
8:             end for
9:         end if
10:        if pickPlace in movements then
11:            for all pickPlaceWins : movements[pickPlace] do
12:                Remove overlapping slide movements
13:            end for
14:        end if
15:        for all action : movements do
16:            if len(movements[action] > 0) then
17:                predActions.insert(GetShape(obj) + action)
18:            end if
19:        end for
20:    end for
21: end procedure
```

Algorithm 3 Task 1: Atomic Action Recognition

procedure AtomicActionRecognition(*scenes*)
    $groundTruthActions \leftarrow list()$
    $predActions \leftarrow list()$
    for all $scene : scenes$ **do**
        $sceneGtActions =$ GetGroundTruth(*scene*)
        $groundTruthActions.insert(sceneGtActions)$
        $actionWindows \leftarrow$ AtomicActionWindows(*scene*)
        $scenePredActions \leftarrow$ AtomicActionPredictions($actionWindows$)
        $predActions.insert(scenePredActions)$
    end for
    $mAP \leftarrow$ CalculatemAP($groundTruthActions, predActions$)
end procedure

4.1.4 Rule Based System

We saw in Fig. 4.3 that different actions have distinct object trajectories. Using DeepSORT we obtained the trajectories of each of the objects in the scene. This can be fed into a handcrafted rule based system to classify the actions.

Algorithm. 3 shows the pseudo-code for classifying the atomic actions as defined in Task 1 of the CATER dataset. It first identifies all the windows in which the location of the object changes. The windows are then segregated into different buckets based on the type of motion that was identified. If the location of the object changes just along the x and y axis, it indicates a slide action whereas if z location of object changes followed by changes in x, y coordinates, it denotes a pick-place action. The same reasoning can be extended to a 2-D image where we just have x and y coordinates.

The rules are designed to be flexible as all observations are made for a window of 10 frames so that it provides some room for error during the object tracking phase. Algorithm .3 uses the procedures defined in Algorithm. 1 and Algorithm. 2 for getting the atomic action windows and atomic action predictions of a scene respectively.

It must be noted that we omit the rotate action while experimenting with hand-crafted rules as the physical location of the object doesn't change in case of rotation.

Algorithm. 6 extends Algorithm. 3 and is used for identifying composite actions. Using the first algorithm, the start and end time for each of the actions can be estimated with some error. These action windows can then be used to define a temporal relationship between pair of actions. Algorithm .6 uses the procedures defined in Algorithm. 1, Algorithm. 4 and Algorithm. 5 for getting the atomic action windows, composite action windows and composite action predictions of a scene respectively.

4.2 R3D

This method is based on the Two stream Inflated 3D ConvNet (I3D) (Carreira and Zisserman (2018)) network. The architecture extends the 2D ConvNet inflation to 3D for learning spatio-temporal features from the videos. Fig. 4.4 shows the inflated Inception-V1 and its corresponding Inception module.

The model is trained using pre-trained weights from Kinetics dataset (Kay *et al.* (2017)) and uses ResNet as its base. We performed multiple experiments by varying the duration of the clip and the sampling rate of the video. Section 5.2 reports the results obtained using this architecture.

Algorithm 4 Get Composite Action Prediction Windows

```
1: procedure CompositeActionPredictionWindows(sceneActionWindows)
2:     actionWindows ← list()
3:     for all obj : sceneActionWindows do
4:         movements ← sceneActionWindows[obj]
5:         if contain in movements then
6:             for all containWins : movements[contain] do
7:                 Remove overlapping pickPlace and slide movements
8:             end for
9:         end if
10:        if pickPlace in movements then
11:            for all pickPlaceWins : movements[pickPlace] do
12:                Remove overlapping slide movements
13:            end for
14:        end if
15:        for all action : movements do
16:            actionName ← GetShape(obj) + action
17:            if len(movements[action] > 0) then
18:                actionWindows[actionName].insert(movements[action])
19:            end if
20:        end for
21:    end for
22: end procedure
```

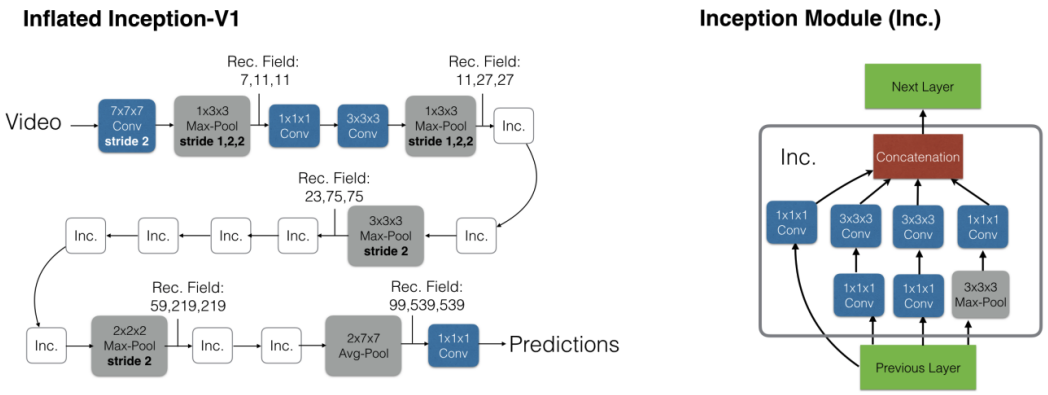


Figure 4.4: Architecture for I3D network (Source: Quo Vadis, Action Recognition? A New Model and the Kinetics Dataset (Carreira and Zisserman (2018))). The model inflates the 2D ConvNet architecture so that it can be used for videos.

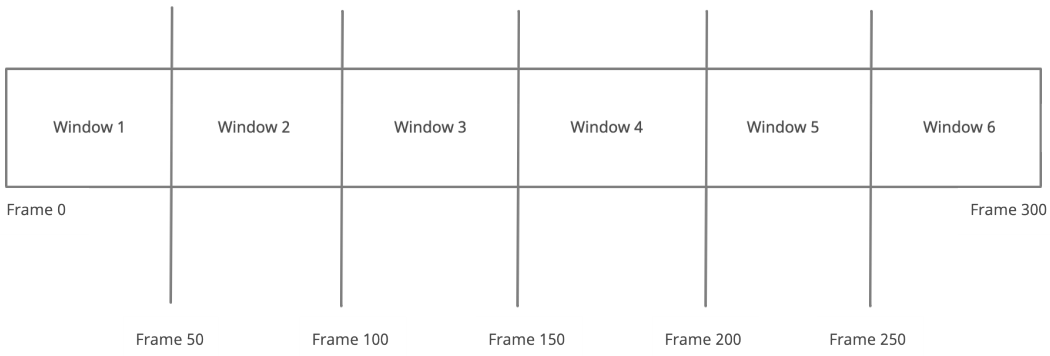


Figure 4.5: Window Based Training technique: The whole video is divided into multiple windows and training is performed to learn atomic actions for each windows

Algorithm 5 Get Composite Action Predictions

```
1: procedure CompositeActionPredictions(actionWindows)
2:     compositeActions ← list()
3:     for all action_1 : actionWindows do
4:         for all action_2 : actionWindows do
5:             windows_1 ← actionWindows[action_1]
6:             windows_2 ← actionWindows[action_2]
7:             for all win_1 : windows_1 do
8:                 for all win_2 : windows_2 do
9:                     rel ← GetTemporalRel(win_1, win_2)
10:                    compositeActions.insert(action_1 + rel + action_2)
11:                end for
12:            end for
13:        end for
14:    end for
15:    compositeActions ← list(set(compositeActions))
16: end procedure
```

4.2.1 Window Based Method

The performance of I3D for learning the atomic actions in the videos was quite encouraging and it provided the motivation to extend this approach for the identification of composite actions. A simple training technique was devised by dividing the whole video into multiple windows and learning atomic actions for each of the windows. The CATER (Girdhar and Ramanan (2019)) dataset provides annotations indicating the start and end frames for each of the atomic actions. A script was

Algorithm 6 Task 2: Composite Action Recognition

```
1: procedure CompositeActionRecognition(scenes)
2:     groundTruthActions ← list()
3:     predActions ← list()
4:     for all scene : scenes do
5:         sceneGtActions = GetGroundTruth(scene)
6:         groundTruthActions.insert(sceneGtActions)
7:         atomicActionWins
            ← CompositeActionPredictionWindows(scene)
8:         compositeActionWins
            ← CompositeActionPredictionWindows(atomicActionWins)
9:         scenePredActions
            ← CompositeActionPredictions(compositeActionWins)
10:        predActions.insert(scenePredActions)
11:    end for
12:    mAP ← CalculatemAP(groundTruthActions, predActions)
13: end procedure
```

used for generating atomic action labels for the specified window using the available annotations. The script made is quite easy to setup new experiments with different number of windows. Fig. 4.5 shows how the video is divided into multiple windows for training purpose. For eg., if the number of windows is set to 6, then the video will be divided into 6 equal parts and each window will be treated as a independent sample. During training, the labels generated using the script would be used for learning the atomic actions for that particular window. Section 5.2.1 describes the experiments in detail.

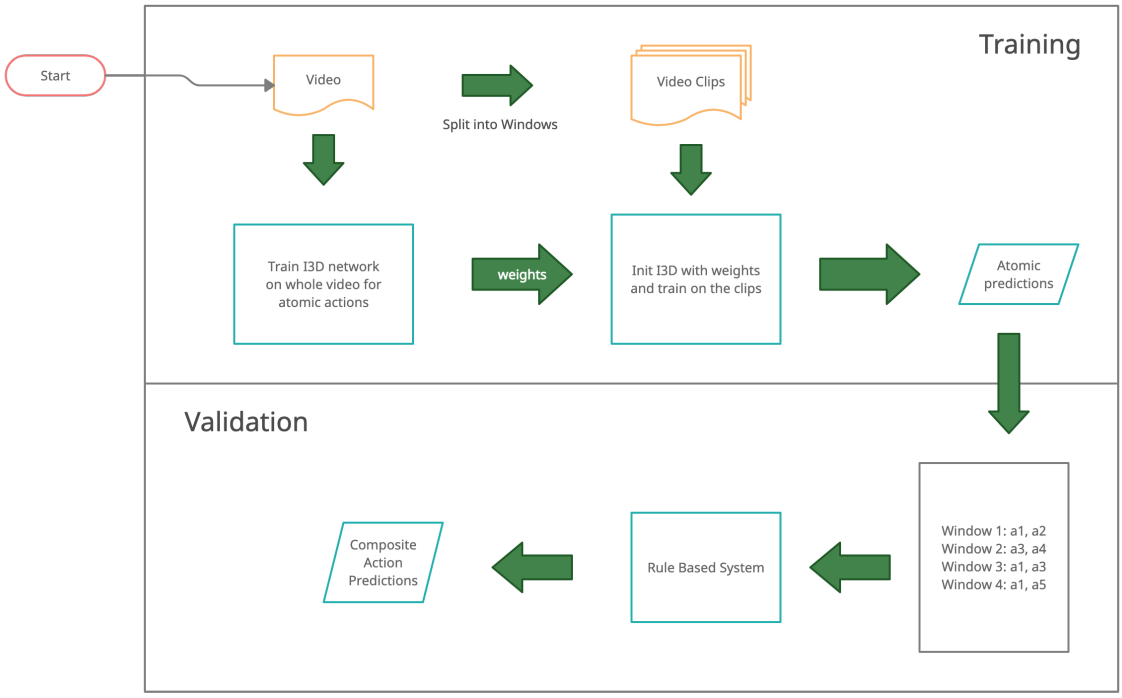


Figure 4.6: Window Based Method Architecture: The figure shows the training and validation process used for Window Based Method

During inference, clips from each of the windows are passed through the network to obtain atomic action labels for the windows. Once the atomic action labels are obtained, action's start and end frame is set to be the same as the window's start and end frame. Fig. 4.6 shows the end to end training and validation architecture used for this method. Finally, all the action windows $(action, start, end)$ is passed through a simple rule based system to determine the composite action labels. Algorithm 6 describes the rule based system used for determining the composite actions based on predictions from all of the windows.

Initially, experiments were performed using number of windows from 6 to 12. As described in Section 5.2.1, 6 windows performed the best amongst these. Next, instead of taking non-overlapping windows from the videos, random clips of a fixed

sequence length were picked from the video. A pre-determined number of samples were extracted from each of the videos. The original script was modified to allow generation of random clips. Using random clips massively improved the availability of training data. The number of samples from each video was set to 20 so that enough number of samples can be extracted from each video. Also, the sequence length was set to 50 since in the previous experiment 6 windows (i.e. 50 frames per window) performed the best.

4.3 Non Local Net

This method is based on the Non-local neural networks(Wang *et al.* (2018)) which uses non-local operations for capturing long term dependencies. The non-local operation calculates the weighted sum of features at all positions instead of using just the neighbouring positions. In our experiment, we replace the final two Conv3D layers of the R3D model with non-local blocks. Similar to R3D we perform multiple experiments by varying the clip duration and sampling rate. Section 5.3 reports the results obtained using this architecture.

4.4 Transformers

In this thesis, multiple transformer based approaches were used for experimentation to learn the atomic actions in the videos. The experiments didn't perform well and failed to effectively identify the atomic actions in the videos. This section describes the network architectures for the different transformer based approaches.

First, a transformer network inspired from Video Action Transformer Network (Girdhar *et al.* (2019)) was used for task 1 and task 2. As described in the original paper, I used a I3D(Carreira and Zisserman (2018)) network as the base for extracting feature maps from raw video frames. Next, a Faster-RCNN was used on the first frame

of the video to extract the bounding boxes of all the objects. The bounding boxes along with the feature map was passed to a ROI-align layer to get output feature maps. Roi-align was originally proposed in the Mask R-CNN paper (He *et al.* (2018)) which takes a list of bounding boxes along with the input feature map and returns a list of output feature maps of the same dimension. Finally, the output feature maps are concatenated along with W dimension and pass it through a linear layer to get the query (q) vector. Also, the original feature map obtained from I3D is passed through two independent linear layers to get the key and value (k, v) vectors. The (q, k, v) vectors are passed through 3 layer multi-head attention network(Vaswani *et al.* (2017)) to get the final output. The final output is then passed through a fully connected layer to get the class predictions. Section 5.4 describes the results obtained using this network.

Next, ViViT, a video vision transformer (Arnab *et al.* (2021)) was used for experimenting with task 1. ViViT is a pure transformer based model for video classification. It builds on top of ViT: Vision Transformer (Dosovitskiy *et al.* (2021)) which is a pure transformer based network for classifying images. Fig. 4.7 shows the network architecture for ViViT. The network first uses a standard ViT network to extract encodings from each of the frames and then the class tokens of these encodings are passed to a temporal transformer for learning the temporal attention encodings. Finally, the output of the temporal transformer encoder is passed through a MLP head for learning the video action class. The ViT network takes a input frame and divides it into non-overlapping patches. Vector embeddings are then obtained from each of these patches using a linear layer and the embeddings from all the patches are stacked to get the embedding tokens. These tokens are of shape $(BxNxd)$ where B is the batch size, N is the number of tokens and d is the dimensions of the tokens.

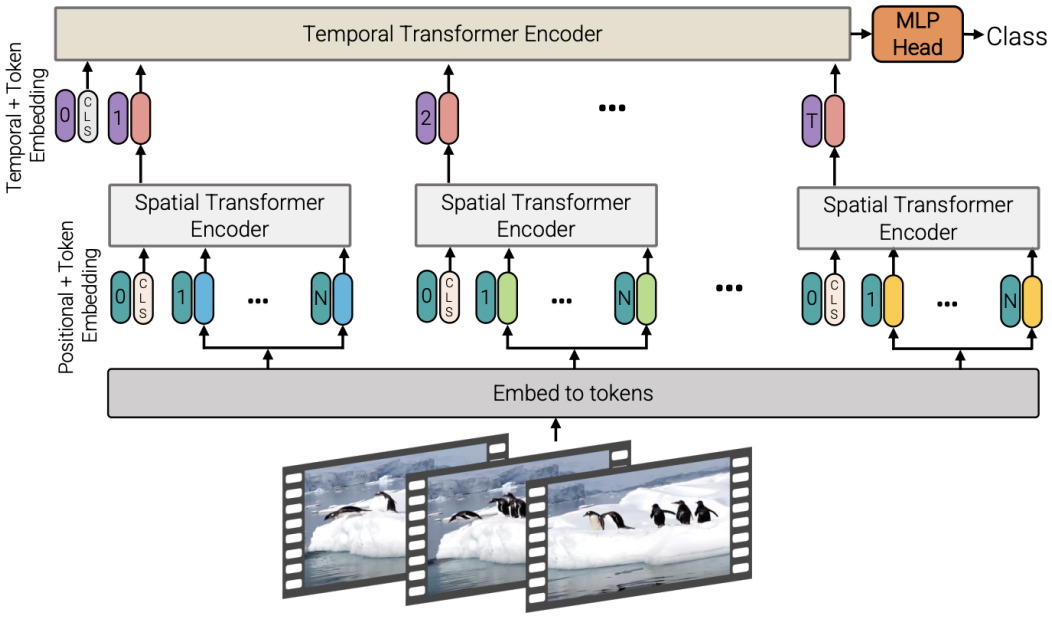


Figure 4.7: Architecture for ViViT: A Video Vision Transformer (Source: Fig 4. in Arnab *et al.* (2021))

Chapter 5

RESULTS

In this chapter the results of various experiments are discussed. For rule based system, precision, recall and accuracy is reported and for other deep learning approaches, mAP(mean average precision) is used as the performance metric.

5.1 Rule Based System

This section discusses the performance of different components of the rule based system.

5.1.1 Object Detection

Multiple experiments were performed for object detection by changing the number of input channels, performing data augmentation and by using pre-trained weights. Table. 5.1 shows the results of the different experiments. Fig. 5.2 visualizes the mAP using different experimental setups. Using pre-trained weights from COCO Lin *et al.* (2015) dataset improved the mAP significantly and using pre-trained weights from LaCATER Shamsian *et al.* (2020) achieved the best results. Fig. 5.1 shows the progression of loss function for the different experiments that were performed. It can be noticed that using pre-trained weights not only improved the mAP, but also helped the network in learning faster. The value for loss decreases sharply when COCO or LaCATER's pre-trained weights were used.

Fig. 5.3 shows a few sample frames annotated with bounding boxes and their corresponding class labels. It can be noticed that the network is able to draw correct bounding boxes even for very small objects in the scene. It is able to correctly detect

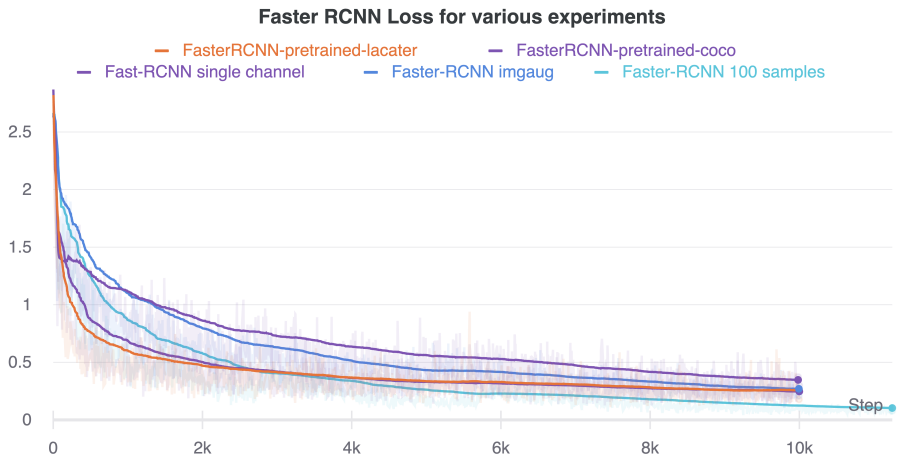


Figure 5.1: Comparison of Faster-RCNN Loss for Different Experiments: The plot shows how the loss decreases after each training step. It can be noticed that the loss for the network initialized using pre-trained weights from LaCATER dataset (orange curve) decreases the fastest whereas the network initialized with pre-trained weights from COCO dataset (purple curve) takes the most amount of time to converge. The plots make it clear that not only does the use of pre-trained LaCATER weights improve the performance, it also speeds up how fast the learning happens.

almost all objects in the sample images.

5.1.2 Object Re-identification

As described in Section. 4.1.2, Siamese Net was used for distinguishing between objects in the scene. The training was done for 40 epochs using 300 images from each of the 192 classes. Table 5.2 shows the training and testing results. The network achieved very high testing accuracy of over 99%. Fig. 5.4 shows the plot for the loss function during training.

The object detection module was used for obtaining frame wise bounding boxes and class labels for all videos in the validation set. The bounding boxes along with the

Method	AP[IoU=0.5]	AP[IoU=0.75]	AR[IoU=0.5:0.95]
RGB images	92.9	87.1	72.1
RGB with image augmentation	94.0	84.2	74.5
Grayscale images	94.7	84.2	74.3
Grayscale pretrained on COCO	97.3	90.8	78.6
Grayscale pretrained on LaCATER	**99.5**	**96.3**	**82.0**

Table 5.1: Comparison of average precision values for different experimental settings using Faster-RCNN. The network depicted the best results when grayscale images were passed through the Faster-RCNN network pretrained on a LaCATER Dataset. The results clearly show that use of LaCATER weights results in significantly better performance as compared to what is obtained using COCO weights.

trained object re-identification module was passed to the DeepSORT model to obtain trajectories for all objects in the scene. The results were stored as CSV files which were later fed into a handcrafted rule based system to obtain action predictions.

5.1.3 Rule-based System

The preceding sections described how object detection, re-identification and tracking was done to obtain trajectories for all objects in the scene. Before the trajectories obtained from DeepSORT are used for action classification, we test our handcrafted rules using the annotations from the training data. The training data contains frame wise locations for each of the objects in the scene.

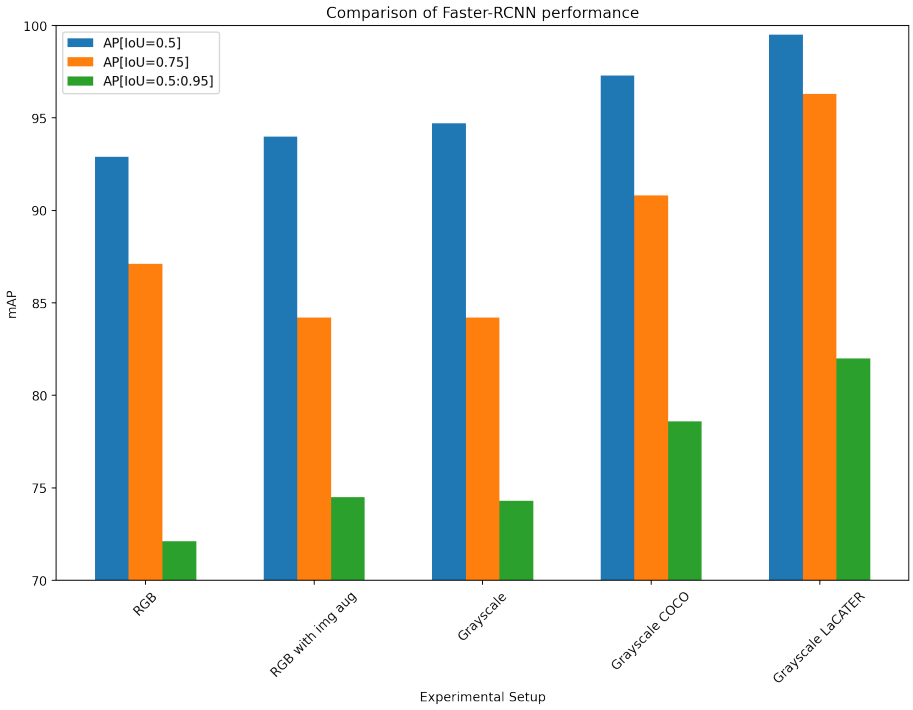


Figure 5.2: Comparison of Faster-RCNN Performance: The plot shows the performance obtained using Faster-RCNN network under different experimental settings. Clearly using Grayscale images pre-trained on LaCATER dataset performed the best amongst these experiments.

For task 1, we pass these coordinates through our rules as described in Algorithm 3. Table A.1a shows the class wise precision and recall using the training data. It can be seen that the rules are able to accurately classify almost all actions. It must be noted that we have reported precision and recall separately for this experiment as the recall doesn't vary based on different values of precision. The results beat the state of the art performance reported in the CATER dataset paper by about 2%. Table A.1b shows the class wise performance when object trajectories are used instead of training data. The performance degrades considerably, indicating that the trajectories obtained using DeepSORT are not accurate. The objects in the

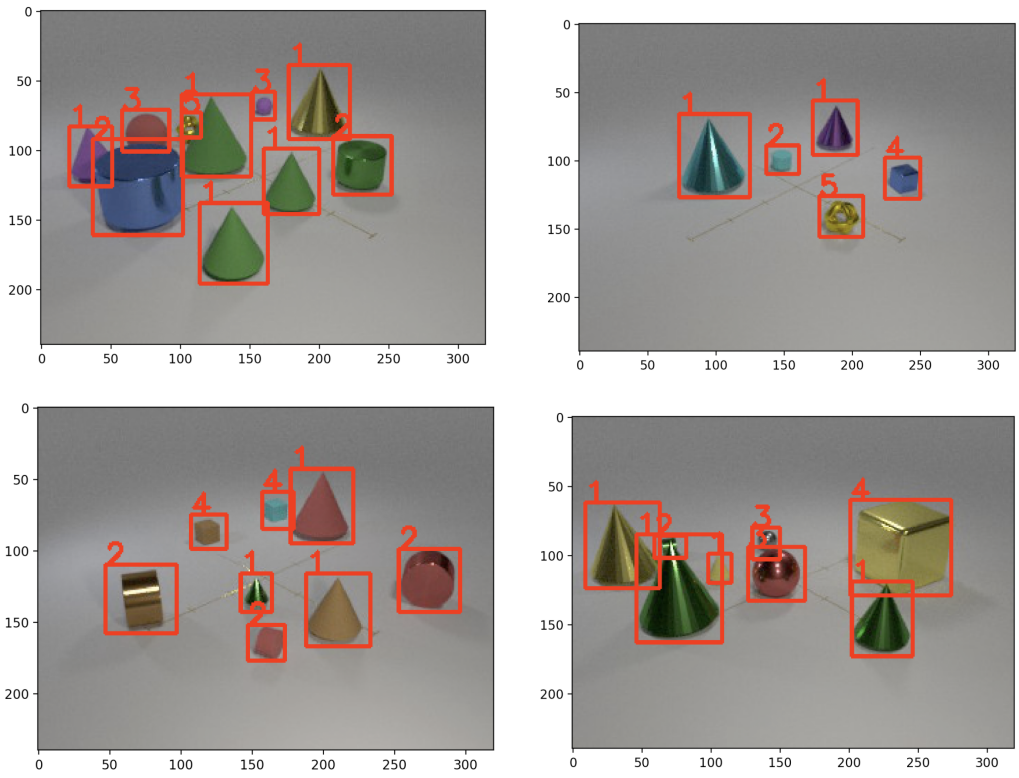


Figure 5.3: Object Detection Results: The images show the predicted bounding boxes and the class labels on some samples frames of a video in the CATER dataset. Faster-RCNN network pre-trained on LaCATER dataset was used for these predictions and the plots show that the network is able to draw a fairly accurate bounding box even for smaller or partially occluded objects in the frame.

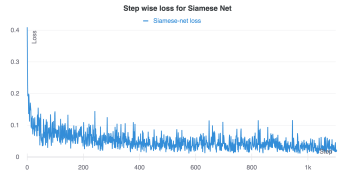


Figure 5.4: Siamese Network Loss

Metric	Value
Training Loss	0.022
Training + Val accuracy (20% unseen samples)	99.99%
Correct/Total	57599/57600

Table 5.2: Results from Siamese Network

scene undergo long term occlusion and containment which would lead to tracks being deleted and reinitialized.

Similarly, for task 2, Algorithm 6 is fed with the coordinates obtained from tracking to predict composite actions. Again, to validate the rules, the training data is used for action predictions and Table A.2 shows the class wise scores for the first 15 actions. As seen for task 1, the results using object trajectories didn't perform well since the tracks were not accurate. Table A.3 shows the results for first 15 actions using trajectories obtained from DeepSORT.

Table 5.3 shows the overall performance of rule based systems for both tasks using training data and tracking information. The performance of the handcrafted rules using training data beats the state of the art for both task 1 and task 2 but the results are not replicated when training data is replaced with tracking information. There's a definite possibility for improvement using a robust tracker that handles long term occlusion and containment efficiently.

5.2 I3D

Table 5.4 shows the performance of using R3D network for task 1 and Table 5.5 shows the performance for task 2. For task 1 using clips with 64 frames sampled uniformly at a rate of 4 performed better as compared to clips with 32 frames samples at a rate 8. The model was trained for 150 epochs and the experiment using clips with 64 frames achieved a mAP of 97.2% which is very close to the state-of-the-art performance. For task 2 R3D with 64 frames achieved an mAP of 40.38% which is close to what the original paper states.

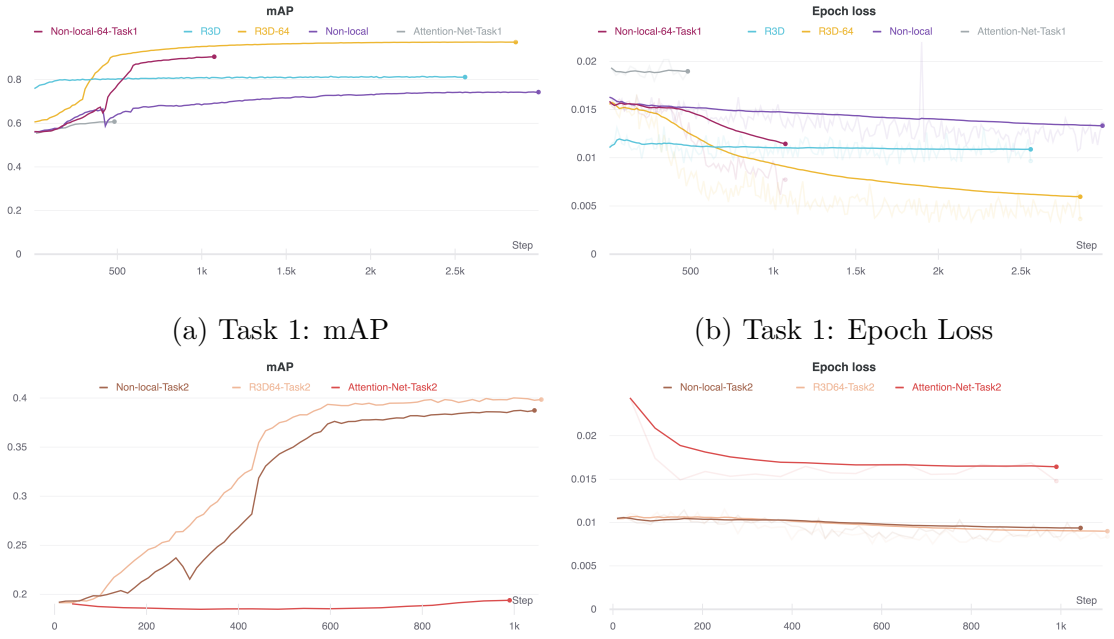
Method	Precision (%)	Recall (%)	F1 Score (%)	Accuracy (%)
Atomic Actions using training data	99.37	99.3	99.33	99.27
Atomic Actions using tracking results	72.5	68.5	70.4	68.6
Composite Actions using training data	71.9	90.1	80.0	92.0
Composite Actions using tracking results	32.0	46.0	38.0	73.0

Table 5.3: Performance of Rule Based System for different experimental settings. The first two rows in the table show the results for atomic action recognition and the next two rows show the results for composite actions. The results clearly show that the the rule-based system performs quite well when coordinates from the training data is used whereas when coordinates from the tracking results are used, the performance drops drastically.

5.2.1 Window Based Method

Window based training strategy was used for learning atomic actions from each of windows and these were used for inferring the composite action labels. mAP metric was used for evaluating the performance of this method.

Calculating mAP: The mean Average Precision (mAP) was calculating by taking a mean of Average Precision (AP) values for all the classes. For calculating the AP, the area under the curve was calculated for the precision vs recall plot. Different



(a) Task 1: mAP

(b) Task 1: Epoch Loss

(c) Task 2: mAP

(d) Task 2: Epoch Loss

Figure 5.5: mAP and Epoch loss plots for R3D, Non-Local Net and Transformers

values of threshold were used for classifying a class as positive label to get the action predictions. Using the predictions at different threshold values, precision and recall values were calculated. Using these precision and recall values, the plot was drawn for precision vs recall.

Table 5.5 shows the performance of window based method for task 2. It surpasses the state-of-the-art performance for task 2 as reported in the CATER (Girdhar and Ramanan (2019)) dataset paper. The results are quite encouraging as 66.57% mAP is significantly higher than the best performing network described in the paper. These results were achieved using random sequences of 50 frames with 20 samples from each video used for training. During inference, clips from all the windows of the video were passed through the I3D network to obtain the atomic action labels and then the composite action labels were determined using the rule based system as described in

Method	mAP(%)
Static R3D + NL	98.9
Rule-Based System using training data[1]	**99.37**
Rule-Based System using tracking results[1]	72.5
R3D with 32 frames	81.1
R3D with 64 frames	97.2
Non-local net with 32 frames	74.3
Non-local net with 64 frames	90.4
Transformer Network	60.32

Table 5.4: Comparison of mAP using different approaches for atomic action recognition with static camera setup. The first row lists the state-of-the-art performance from the CATER dataset.

Algorithm 6.

Static Camera

Initially, experiments were performed with a static camera setting where the camera position was kept the same for the entire duration of the video. Experiments using a static camera setting performed much better than a moving camera setting. Results for moving camera is described in 5.2.1.

As described in Section 4.2.1, multiple experiments were performed using different number of windows. Table 5.6a shows the performance achieved for different experimental setups. It can be clearly seen that amongst using 4-12 windows, 6 windows (i.e. 50 frames per window) performed the best. Fig. 5.8 shows the comparison of validation mAP and Epoch loss for different number of windows. Moreover, Fig. 5.7

Method	mAP(%)
R3D + NL + LSTM (64 frames)	53.1
Handcrafted rules using training data[1]	**71.9**
Handcrafted rules using tracking results[1]	32.0
R3D with 64 frames	40.38
Non-local net with 64 frames	38.63
Transformer Network	19.28
I3D With random sequences of 50 frames	66.47

Table 5.5: Comparison of mAP using different approaches for composite action recognition with static camera setup. The first row lists the state-of-the-art performance from the CATER dataset.

compares the performance of the experiments performed with the baseline performance.

The results from using different number of windows showed that using 6 windows clearly outperformed other window sizes. It provided the motivation to use sequences of length 50 for the random training approach. Fig. 5.9 shows the comparison of of validation mAP and Epoch loss for Random sequences vs 6 Windows. It can clearly be seen that the validation mAP using random window surpasses the performance using 6 windows. The same is reasserted by the epoch loss where the epoch loss using random windows consistently decreases as training continues whereas using 6 windows, the loss starts increasing after a certain amount of time. These plots reaffirm the importance of having a large number of training samples available for the network to learn effectively.

(a) Task 1: mAP

(b) Task 1: Epoch Loss

Figure 5.6: Comparison of validation mAP and Epoch loss for 6 Windows with and without pre-trained weights: The plot on the left clearly shows that using weights from the whole video (pink curve) results in the fastest convergence and also improves the performance of the network significantly.

Moreover, experiments were performed to see the effect of using pre-trained weights. Table 5.6b shows the performance for task 2 with and without pre-trained weights for 6 windows. In one of the experiments, the weights were randomly initialized and in the next experiments weights from the I3D network trained on Kinetics dataset was used. In the final experiment, weights of the I3D network trained on the full video of the CATER (Girdhar and Ramanan (2019)) dataset was used. It can be clearly seen from the results that using pre-trained weights from the whole video helps in improving the learning capability of the network. Fig. 5.6 shows the comparison of validation mAP and Epoch loss for 6 Windows with and without pre-trained weights. Encouraged by the results, pre-trained weights from the whole video was used for all the other experiments. Moreover, Fig. 5.10 shows the same results as a bar plot for easier visualization.

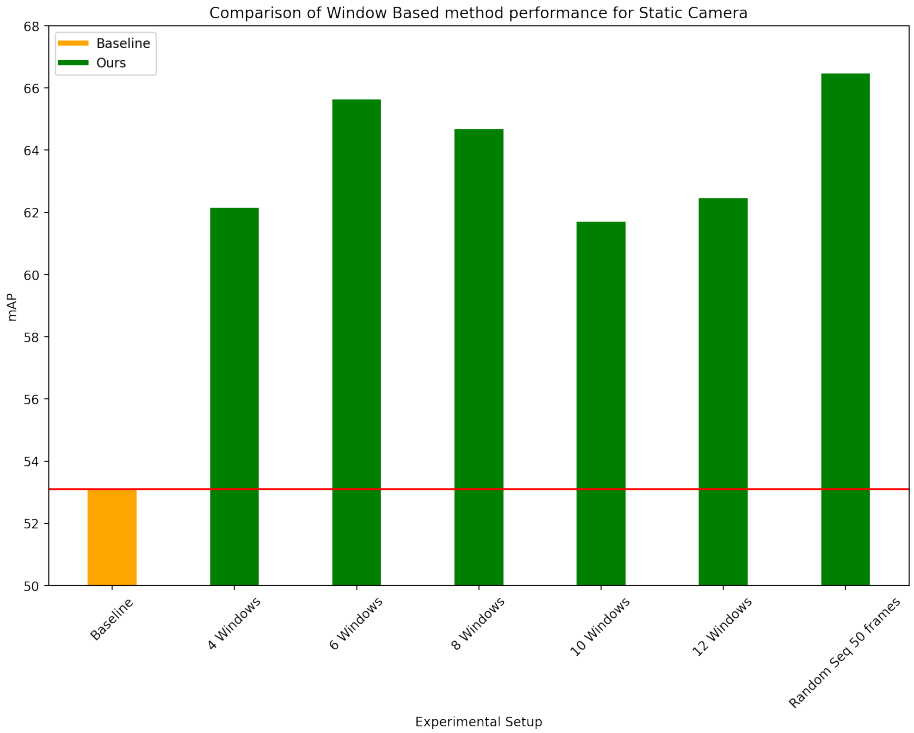


Figure 5.7: Plot of Window Based Method Performance for Static Camera: The plot compares the performance of the window based method for static camera setting with the baseline performance. The baseline performance is shown in orange and all the other bars show the results of the experiments performed in this thesis.

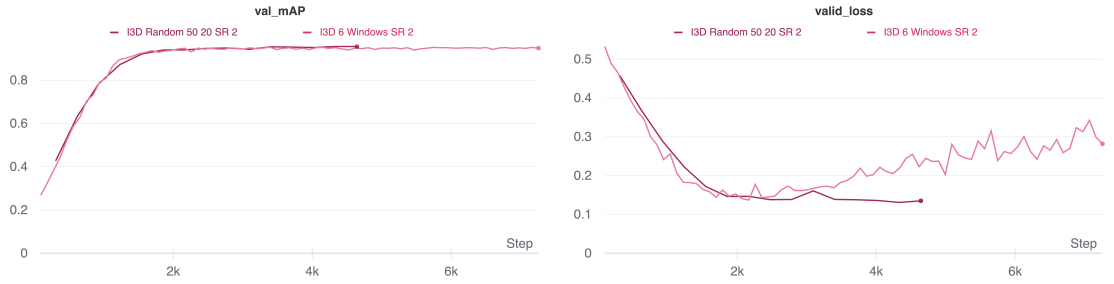
Moving Camera

After performing numerous experiments in a static camera scenario, a few experiments were performed for a moving camera setup. With a moving camera, all the experiments need not be repeated as the experimental setup is very similar. For eg., experiments in the previous section (Section 5.2.1) indicated that using pre-trained weights from the whole video aids in learning the actions on smaller clips. This learning was utilized for the moving camera approach and all the experiments were performed by initializing the model with weights from the whole video.

(a) Task 1: mAP

(b) Task 1: Epoch Loss

Figure 5.8: Comparison of mAP and Loss for Different Number of Windows: Multiple experiments were performed using different window sizes to find the optimal window size that gives the best result for the task of composite action recognition. The results indicate that using random clips with 20 samples taken from each video outperforms other experimental settings



(a) Task 1: mAP

(b) Task 1: Epoch Loss

Figure 5.9: Comparison of Map and Loss for Random Vs 6 Windows: The plot on the right shows that when random windows are used(dark pink curve), the validation loss steadily decreases whereas using 6 windows (light pink curve), the validation loss starts increasing after around 5000 steps.

Method	mAP(%)
I3D With 4 Windows	62.15
I3D With 6 Windows	65.63
I3D With 8 Windows	64.67
I3D With 10 Windows	61.7
I3D With 12 Windows	62.46
I3D With Random Sequences of 50 frames	**66.47**

(a) Experiments using different window sizes

Method	mAP(%)
I3D With 6 Windows random weight initialization	62.36
I3D With 6 Windows pre-trained with weights from whole video	**64.67**
I3D With 6 Windows pre-trained on Kinetics dataset	61.37

(b) Experiments with and without pre-trained weights

Table 5.6: Comparison of mAP for Composite Actions for Different Experimental Setups: (a) The table shows the performance of the I3D network when different number of windows are used. The results indicate that using random clip sequences of size 50 with 20 samples taken from each video outperforms other experimental settings (b) The table shows the performance of the I3D network with and without pre-trained weights. The results clearly indicate that using weights from the whole video helps in improving the performance in terms of mAP.

Moreover, experiments using different window sizes, showcased a similar performance with 6-10 windows performing quite well. Therefore, for moving camera experiments were performed only for these window sizes. Table 5.7a compares the performance using window based training technique to the best baseline performance listed in the CATER dataset. Table 5.7b shows the performance obtained when different window sizes were used. The table clearly shows that the best performance was obtained using 10 windows. It must be noted, that with a static camera setting,

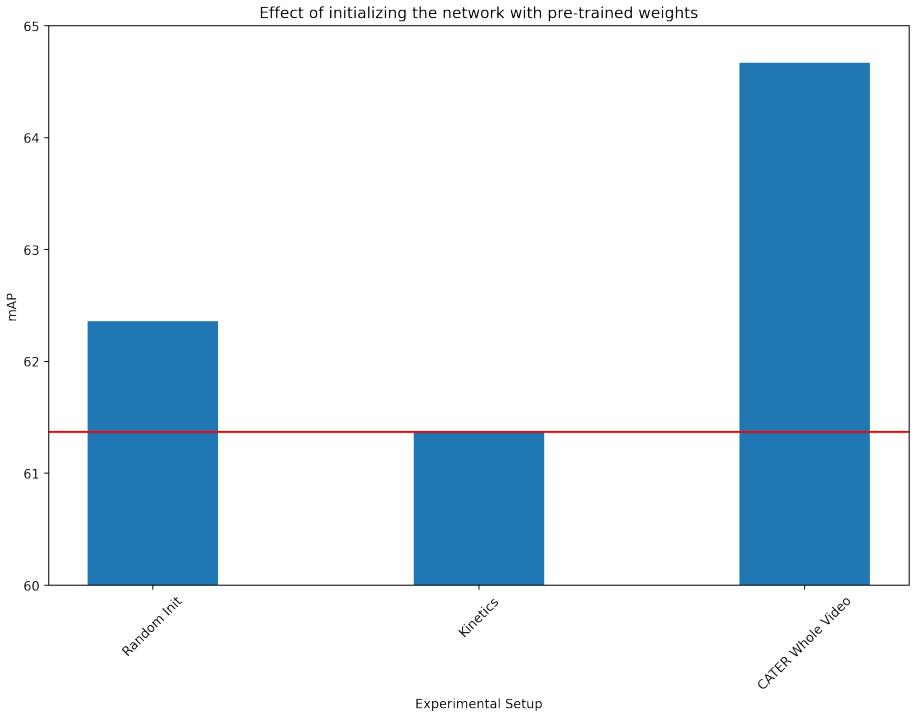


Figure 5.10: Plot of Window Based Method Performance for Different Weight Initialization

the best performance was obtained using random sequences of 50 frames which is equivalent to 6 windows in terms of clip duration. Fig. 5.12 shows the comparison of validation mAP and epoch loss for different number of windows. Moreover, 5.11 shows the comparison of mAP for different experimental setups using moving camera.

5.3 Non Local Network

Table 5.4 shows the performance of using Non-local neural network for task 1 and Table 5.5 shows the performance for task 2. Similar to R3D, for task 1 experiments were conducted using 64 frames and 32 frames. Network with 64 frame clips performed better as compared to the one with 32 frames. For task 2, the network performed poorly and couldn't achieve the performance as stated in the original CATER paper.

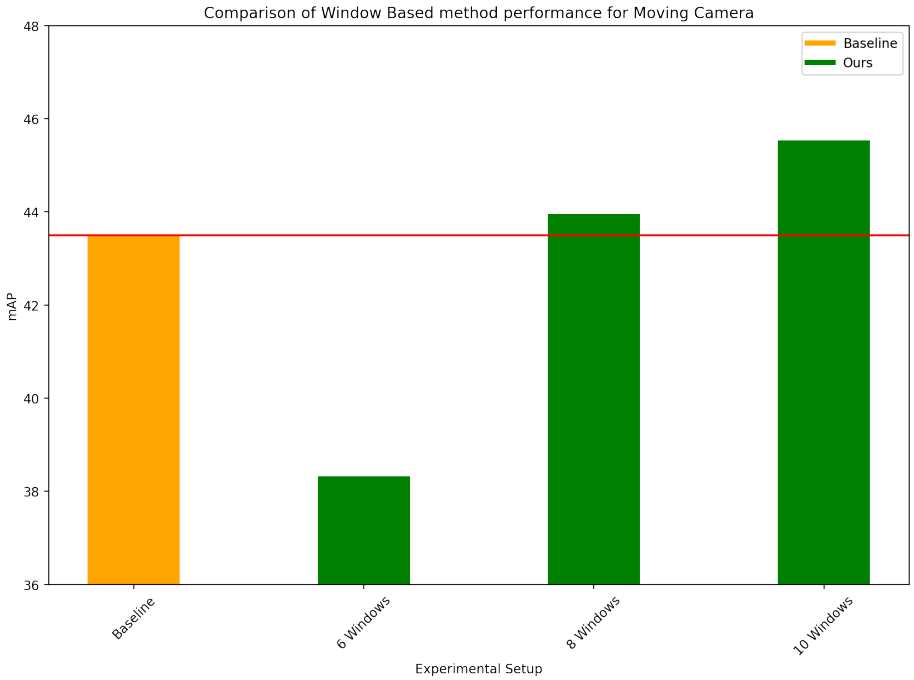
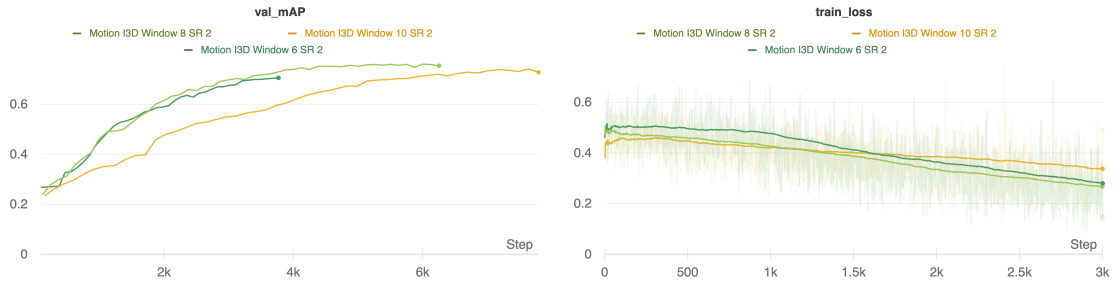


Figure 5.11: Plot of Window Based Method Performance for Moving Camera: The plots show the baseline performance in orange and the green lines show the results of the experiments performed in this thesis.



(a) Task 1: mAP

(b) Task 1: Epoch Loss

Figure 5.12: Comparison of validation mAP and Epoch loss for different number of windows using videos containing camera motion: The plot on the left clearly shows that using 8 windows (light green curve) results in the best validation mAP and its performance is significantly better than using 10 windows (orange curve).

Method	mAP(%)
R3D + NL + LSTM (32 frames)	43.5
I3D With 10 windows	45.53

(a) Task 2: Composite Action Recognition

Method	mAP(%)
I3D With 6 Windows	38.32
I3D With 8 Windows	43.95
I3D With 10 Windows	45.53

(b) Different window sizes

Table 5.7: Comparison of mAP using different approaches for moving camera: (a) The first row lists the baseline performance as mentioned in the CATER dataset. Using I3D with 10 windows is able to outperform the mAP by around 2%. (b) The table shows the mAP obtained using different number of windows. It can be clearly seen that using 10 windows results in the best performance. Moreover, using 8 windows also surpasses the baseline mAP by a very small margin.

It is evident that using a larger sample duration lets the model learn more efficiently as the spatio-temporal features are captured in much more detail.

5.4 Transformers

The results using transformer network is listed for the sake of completeness but there would be future iterations to this network that should improve its performance. Table 5.4 shows the existing performance for task 1 and Table 5.5 shows the performance for task 2.

Fig. 5.5a and 5.5b shows a comparison of mAP and epoch loss for R3D, Non Local nets and transformers for task 1. Similarly, Fig. 5.5c and 5.5d shows a comparison of mAP and epoch loss for R3D, Non Local nets and transformers for task 2.

[1]For handcrafted rules all actions except actions containing rotation are considered, since rotation doesn't have any translational motion

Chapter 6

CONCLUSION AND FUTURE WORK

6.1 Conclusion

In this thesis, the first step was taken towards effectively identifying composite actions in videos under long-term occlusion and containment. The experiments were performed on the CATER dataset which is a synthetic dataset containing simple objects performing actions such as slide, pick & place, rotate and contain. A controlled environment was chosen for the experiments so that it is easier to analyze the results of the experiments and identify the best approach for solving the problem of composite action recognition.

In this thesis, a rule based system was proposed for identifying atomic and composite actions in videos. The algorithm was validated using frame level annotations provided with the training data and it performed exceedingly well for both Task 1 and Task 2. It achieved a mAP of 99.37% for Task 1 and 71.9% mAP for Task 2. It provided the motivation to use a multi-object tracking technique for predicting the coordinates of all the objects in each of the frames and pass it to the rule based system. Predicting the coordinates of all the objects in the scene required 3 key tasks, i.e. object detection, object re-identification and multi object tracking. A Faster RCNN network was trained on using a few hundred samples and it achieved a mAP of 96.3% for 0.75 IOU. Moreover, a Siamese network was trained on a few thousand samples for object re-identification and it achieved 99.99% accuracy when all the training and validation images were used. Finally, a DeepSORT algorithm was used for object tracking. It is evident that tracking based approach doesn't achieve

the desired accuracy as the objects in the scene undergo long term occlusion and containment. The DeepSORT based tracker makes multiple identity switches and isn't able to track the objects accurately. Even though the handcrafted rule based system exceed the baseline accuracy, the rules do not perform well when fed with tracking information.

Experiments using I3D and Non-local neural networks perform well and matches the performance that was stated in the CATER dataset's paper for Task 1. The performance of I3D network on Task 1 motivated to extend it for Task 2. A window based training technique was devised where the whole video was divided into multiple non-overlapping clips and labels where generated for each of those clips using a custom script. Multiple experiments were performed using this approach and the use of random sequences of 50 frames with 20 samples picked from each video, performed the best with a validation mAP of 66.47%. This surpassed the mAP reported for Task 2 in the CATER dataset by a good margin. Moreover, the network was initialized with pre-trained weights from the whole video and it provided a performance boost of around 4-5% in terms of mAP. It must be noted that this although approach works quite well for this particular dataset, it might not generalize well for other datasets.

6.2 Future Work

In this thesis, all the experiments were performed under a controlled environment and real-world datasets might differ in terms of complexity of the scene. In future, work must be done to perform experiments using a real-world dataset. To effectively test the performance of the network, it is necessary that the dataset doesn't contain any scene bias so that the network doesn't start learning from the background of the scene instead of focusing on the actual action taking place in it. Moreover, it would be challenging to come up with frame level annotations for the actions being performed

and the coordinates for all the object of interest in the scene. In a synthetic dataset, these annotations were easily made available and could be easily extended to other objects or actions. It would be quite insightful to test the effectiveness of window based training technique on a real dataset. Moreover, experiments can be performed to see if this technique generalizes well for other datasets with pure temporal ordering.

Moreover, in future work must be done to introduce new objects and actions in the videos and test the effectiveness of transfer learning for the new videos. Since, its a synthetic dataset and the code for creating the dataset is open-source it would be quite easy to create new videos with new actions or objects introduced to the scene.

REFERENCES

Arnab, A., M. Dehghani, G. Heigold, C. Sun, M. Lučić and C. Schmid, "Vivit: A video vision transformer", (2021).

Carreira, J. and A. Zisserman, "Quo vadis, action recognition? a new model and the kinetics dataset", (2018).

Donahue, J., L. Anne Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, K. Saenko and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description", in "Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)", (2015).

Dosovitskiy, A., L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit and N. Houlsby, "An image is worth 16x16 words: Transformers for image recognition at scale", (2021).

Girdhar, R., J. Carreira, C. Doersch and A. Zisserman, "Video action transformer network", (2019).

Girdhar, R. and D. Ramanan, "CATER: A diagnostic dataset for compositional actions and temporal reasoning", CoRR **abs/1910.04744**, URL `http://arxiv.org/abs/1910.04744` (2019).

Gu, C., C. Sun, D. A. Ross, C. Vondrick, C. Pantofaru, Y. Li, S. Vijayanarasimhan, G. Toderici, S. Ricco, R. Sukthankar, C. Schmid and J. Malik, "Ava: A video dataset of spatio-temporally localized atomic visual actions", (2018).

He, K., G. Gkioxari, P. Dollár and R. Girshick, "Mask r-cnn", (2018).

He, K., X. Zhang, S. Ren and J. Sun, "Deep residual learning for image recognition", (2015).

Hearst, M., S. Dumais, E. Osuna, J. Platt and B. Scholkopf, "Support vector machines", IEEE Intelligent Systems and their Applications **13**, 4, 18–28 (1998).

Ji, S., 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).

Johnson, J., B. Hariharan, L. van der Maaten, L. Fei-Fei, C. L. Zitnick and R. Girshick, "Clevr: A diagnostic dataset for compositional language and elementary visual reasoning", (2016).

Karpathy, A., G. Toderici, S. Shetty, T. Leung, R. Sukthankar and L. Fei-Fei, "Large-scale video classification with convolutional neural networks", in "CVPR", (2014).

Kay, W., J. Carreira, K. Simonyan, B. Zhang, C. Hillier, S. Vijayanarasimhan, F. Viola, T. Green, T. Back, P. Natsev, M. Suleyman and A. Zisserman, "The kinetics human action video dataset", (2017).

Koch, G., R. Zemel and R. Salakhutdinov, "Siamese neural networks for one-shot image recognition", in "ICML deep learning workshop", vol. 2 (Lille, 2015).

Lin, T.-Y., M. Maire, S. Belongie, L. Bourdev, R. Girshick, J. Hays, P. Perona, D. Ramanan, C. L. Zitnick and P. Dollár, "Microsoft coco: Common objects in context", (2015).

Long, X., C. Gan, G. de Melo, J. Wu, X. Liu and S. Wen, "Attention clusters: Purely attention based local feature integration for video classification", (2017).

Ng, J. Y.-H., M. Hausknecht, S. Vijayanarasimhan, O. Vinyals, R. Monga and G. Toderici, "Beyond short snippets: Deep networks for video classification", (2015).

Ren, S., K. He, R. Girshick and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks", (2016).

Shamsian, A., O. Kleinfeld, A. Globerson and G. Chechik, "Learning object permanence from video", arXiv preprint arXiv:2003.10469 (2020).

Sharma, S., R. Kiros and R. Salakhutdinov, "Action recognition using visual attention", (2016).

Sigurdsson, G. A., G. Varol, X. Wang, A. Farhadi, I. Laptev and A. Gupta, "Hollywood in homes: Crowdsourcing data collection for activity understanding", (2016).

Simonyan, K. and A. Zisserman, "Two-stream convolutional networks for action recognition in videos", (2014).

Tran, D., J. Ray, Z. Shou, S.-F. Chang and M. Paluri, "Convnet architecture search for spatiotemporal feature learning", (2017).

Tzutalin, "labelimg", URL `https://github.com/tzutalin/labelImg` (2015).

Vaswani, A., N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, "Attention is all you need", (2017).

Wang, H., A. Kläser, C. Schmid and C.-L. Liu, "Action recognition by dense trajectories", in "CVPR 2011", pp. 3169–3176 (2011).

Wang, H. and C. Schmid, "Action recognition with improved trajectories", in "2013 IEEE International Conference on Computer Vision", pp. 3551–3558 (2013).

Wang, X., L. Gao, J. Song and H. Shen, "Beyond frame-level cnn: Saliency-aware 3-d cnn with lstm for video action recognition", IEEE Signal Processing Letters **24**, 4, 510–514 (2017).

Wang, X., R. Girshick, A. Gupta and K. He, "Non-local neural networks", (2018).

Wojke, N., A. Bewley and D. Paulus, "Simple online and realtime tracking with a deep association metric", (2017).

Yao, L., A. Torabi, K. Cho, N. Ballas, C. Pal, H. Larochelle and A. Courville, "Describing videos by exploiting temporal structure", (2015).

Yuan, Y., Y. Zhao and Q. Wang, "Action recognition using spatial-optical data organization and sequential learning framework", Neurocomputing **315**, 221–233, URL https://www.sciencedirect.com/science/article/pii/S092523121830849X (2018).

Zheng, Z., G. An, D. Wu and Q. Ruan, "Global and local knowledge-aware attention network for action recognition", IEEE Transactions on Neural Networks and Learning Systems **32**, 1, 334–347 (2021).

APPENDIX A

COMPARISON OF CLASS WISE PERFORMANCE FOR RULE BASED SYSTEM

In this section, the class wise performance of the handcrafted rule based system is compared for both Task 1 and Task 2. Table A.1a shows the precision and recall for Task 1 using frame wise coordinates from the training data. It can be seen that the algorithm is able to precisely predict all the actions with a very high recall rate. When instead of passing the coordinates from the training data annotations, tracking algorithm is used for determining the coordinates, the performance of the algorithm drops drastically. Table A.1b shows the c lass wise precision and recall using tracking information. It can be seen that the algorithm performs well only for cones and shows a high precision and recll but for other objects, the performance is quite poor. Taking a detailed look at the table indicates that the algorithm still shows a high recall rate for most of the objects but the precision decreases. It indicates that the algorithm is making a lot of false positive predictions.

Action	Precision	Recall
sphere slide	1.00	0.98
sphere pick place	1.00	1.00
spl slide	1.00	0.98
spl pick place	1.00	1.00
cylinder pick place	1.00	1.00
cylinder slide	1.00	0.98
cube slide	1.00	0.98
cube pick place	1.00	1.00
cone contain	1.00	1.00
cone pick place	0.96	1.00
cone slide	1.00	0.99

(a) Using training data

Action	Precision	Recall
cone pick place	0.97	0.99
cone contain	1.00	1.00
cone slide	0.90	0.73
spl pick place	0.24	0.98
spl slide	0.47	0.78
cylinder pick place	0.36	1.00
cylinder slide	0.28	0.95
sphere pick place	0.56	0.98
sphere slide	0.57	0.80
cube pick place	0.39	1.00
cube slide	0.34	0.87

(b) Using tracking information

Table A.1: Comparison of Class Wise Performance for Atomic Actions

Table A.2 shows the class wise performance for Task 2 for some of the classes out of the 301 possible classes using the training annotations. It can be seen that the algorithm performs quite well for most of the classes except actions where two actions are happening simultaneously and the temporal relation between the actions is **during**. Similar to Task 1, it can be seen from Table A.3 that when coordinates from tracking method is passed to the algorithm, the performance drops drastically. Apart from a few classes, most of the classes show very poor performance. Again, it must be noted that the recall is still relatively high but the precision is quite low. which indicates a lot of false positives been predicted by the algorithm. The poor performance can be attributes to two factors. First, the tracks predicted by the tracking algorithm are not accurate and have a lot of breaks in them due to occlusion. With a lack of accurate object coordinate, the algorithm fails to determine the actions taking place. Moreover, the composite action prediction using the algorithm is dependent on atomic action prediction. Since, the atomic predictions were not highly precise, the error was compounded while predicting the composite actions.

Action	P	R
sphere slide before sphere slide	1.00	0.95
sphere slide during sphere slide	0.04	1.00
sphere slide before sphere pick place	0.99	0.97
sphere slide during sphere pick place	0.80	0.97
sphere slide after sphere pick place	0.99	0.96
...	...	...
cone pick place before cone slide	0.87	0.99
cone pick place during cone slide	0.66	0.97
cone pick place after cone slide	0.92	0.99
cone slide before cone slide	1.00	0.94
cone slide during cone slide	0.48	1.00

Table A.2: Comparison of Class Wise Performance for Composite Actions Using Training Data: The class wise performance for composite actions using training data is quite good for most of the classes except the ones where two actions are happening simultaneously. The same results are not replicated when coordinates from the tracking based method is used.

Action	P	R
sphere slide before sphere slide	0.33	0.64
sphere slide during sphere slide	0.02	0.88
sphere slide before sphere pick place	0.33	0.80
sphere slide during sphere pick place	0.08	0.91
sphere slide after sphere pick place	0.36	0.77
...	...	...
cone pick place before cone slide	0.77	0.70
cone pick place during cone slide	0.41	0.72
cone pick place after cone slide	0.78	0.69
cone slide before cone slide	0.69	0.53
cone slide during cone slide	0.42	0.74

Table A.3: Comparison of Class Wise Performance for Composite Actions Using Tracking Information: The results indicate that the performance of the rule-based system using tracking information is poor for most of the classes except some of the classes involving a cone.

APPENDIX B

ANALYZING THE PERFORMANCE OF WINDOW BASED TECHNIQUE FOR DIFFERENT EXPERIMENTAL SETUPS

Table B.1 shows the performance of different experimental setups using the window based training technique. The table shows the precision, recall, F1 score, Accuracy and mAP for these experiments. mAP is the most important performance indicator since it summarizes the score into a single number. At the same time, it is interesting to check the precision and recall scores for different experiments. It can be seen that in terms of recall random sequences performed the best where as the precision of random windows was similar to other experimental setups. Having a higher recall rate positively impacted the mAP score as well. Moreover, it must also be noted that the accuracy score doesn't hold much significance since the positive labels would be sparse and the accuracy score would be boosted by just predicting the negative labels correctly. In other words, if the network fails to identify any of the actions and just classifies everything as **no action**, then too the accuracy score would be relatively high.

Type	Pre-trained	Precision	Recall	F1	Accuracy	mAP
4 Windows	Yes	68.58	87.66	76.96	90.61	62.15
6 Windows	No	70.56	71.93	71.24	89.61	62.36
6 Windows	On Kinetics	67.86	84.44	75.25	90.06	61.31
6 Windows	Yes	69.97	91.79	79.41	91.48	65.63
8 Windows	Yes	69.48	91.5	78.92	91.25	64.67
10 Windows	Yes	65.14	94.19	77.02	89.94	61.7
12 Windows	Yes	67.76	86.55	76.01	90.02	62.46
Random	Yes	69.3	95.12	80.19	91.59	66.47

Table B.1: Comparison of Performance for Different Window Sizes: The table shows the precision, recall, F1 score, accuracy and mAP values for different experimental setups.