Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

Dissertation

Presented to the University of Dublin, Trinity College

in fulfilment of the requirements for the Degree of

Doctor of Philosophy in Computer Science

June 2021

Ego-hand Gesture Recognition in Trimmed and Untrimmed Videos for Interactions in Augmented and Virtual Reality.

Tejo Chalasani

Declaration

I, the undersigned, declare that this work has not previously been submitted as an exercise for a degree at this, or any other University, and that unless otherwise stated, is my own work.

Tejo Chalasani

June 10, 2021

Permission to Lend and/or Copy

Abstract

Hand gestures are used as a way of communication in our daily lives. Using hand gestures to interact with the virtual environment in Augmented and Virtual reality is a natural extension from the real to the virtual scenario. Thus, recognising hand gestures as seen by the head-mounted Augmented and Virtual reality devices termed as ego-hand gestures, is a problem worth exploring.

Deep neural networks recently have been used to solve the problem of ego-hand gesture recognition, as they are known for their robustness to handle various problems posed by ego-hand gesture recognition like varying lighting conditions, background environments, skin colour, ego-motion, motion blur and more. However, they need a large amount of data for training and testing, making data collection and annotation process very laborious. This work proposed a novel data augmentation technique and published a new ego-hand gestures dataset (Green Screen Ego-hand Gesture Dataset) that can reduce the labour intensive data collection process while being able to train generalisable networks.

A new deep neural network architecture that works on trimmed video ego-hand gesture recognition paying specific attention to ego-hands in the images was proposed. This network was trained and tested on various available trimmed ego-hand gesture recognition dataset including the Green Screen Ego-hand gesture dataset we proposed, advancing the state of the art recognition performance on all the tested datasets.

Trimmed video recognition is not applicable in a real-world scenario, since real-world scenario contains untrimmed videos. In contrast to trimmed videos, untrimmed videos contains gestures interspersed with non-gesture images. StAG LSTM, an extension to LSTM framework was proposed to train ego-hand recognition deep neural network on untrimmed videos. This addition reduces the number of heuristics used, which are a standard part of existing methods. A new loss function(IG Loss) that better optimises the network and a new evaluation metric that is more useful than the current metrics for measuring accuracy of untrimmed video recognition is also proposed. StAG LSTM with IG Loss advanced state of the art performance on ego-hand gesture recognition on untrimmed video.

గురుర్బ్రహ్మా గురుర్విష్ణుః గురుర్దేవో మహేశ్వరః
గురుసాక్షాత్తు పరంబ్రహ్మా తస్మై శ్రీగురవే నమః

The teacher himself is the creator,
the sustainer and the destroyer of knowledge.
He is verily the very transcendental divinity,
my reverential salutations to that glorious teacher.

Acknowledgments

The role of a teacher is highly revered in Indian culture. I feel fortunate to have been a disciple of such a teacher, Prof. Aljosa Smolic. This work would not have started or progressed without his supervision. I am grateful for the freedom I got to choose my own path, all the valuable suggestions whenever I was stuck, a general direction to my research and help with technical writing where I struggled the most. I am also thankful for the lessons on work-life balance that Prof. Aljosa Smolic provided and lead by example. My first research submission would have been a disaster if not for the help from Dr Jan Ondrej, who had played the role of a co-supervisor for the first few months of my research.

I am indebted to my PhD lab mates for keeping me company for the past four years, introducing me to Bengali, Chinese, Deutsch, French, Irish, and Mexican cuisines, and cultures. I also thank my Irish lab mates for giving me a glimpse into the Irish culture, help me integrate into the culture of the land that hosted me for the entirety of this research work. I can not but mention Ms Gail Weadick, who had to bear with my one too many hardware order requests, making sure that all the bureaucratic paperwork is done on time and arranged all the social lab gatherings. She was courageous to take on the task of making sure our lab was lockdown ready, week in and week out made sure that the lab visit schedules were maintained and communicated to everyone. If not for her efforts the year 2020 would have been a much harder one to deal with personally.

The NVAIL team in NVidia provided the GPU workstation used in the final phase of the research as a part of a collaboration initiative which made training and testing new networks much quicker. The research work was fully supported by Science Federation of Ireland.

The incessant concern and support from my parents and my brother were a motivation to finish this document on time, without which I am not sure it would have been possible. Finally, the one person who shared both my triumphs and failures during these last four years, my wife, Sahitya. I will be forever thankful to her for supporting my decision to leave a lucrative career and pursue research.

TEJO CHALASANI

University of Dublin, Trinity College
June 2021

Contents

Chapter 1

Introduction

The idea of egocentric vision was put forward in seminal work by Steve Mann [1], as humanistic computing where he proposed a framework to put human body, mind and computing apparatus in tandem with each other to facilitate better memory, safety and new forms of communication. He further explored the idea of wearable intelligent camera system that records the perspective of the wearer giving rise to the field of egocentric vision.

It was further explored as Quality of Life Technology (QoLT) for older people, and the physically challenged by Kanade and Herbert in [2]. As summarised by Kanade and Herbert in [2], novelty detection [3], egocentric activity detection [4, 5], context awareness [6], interactivity [7], ego-hand gesture recognition [8] are some of the directions of research happening in egocentric vision. Though research in ego-hand gesture recognition started in the field of sign language detection, it has evolved to the field of user interactions [9, 10, 11, 12]. With the advent of consumer-grade Augmented Reality(AR) and Virtual Reality(VR) devices, ego-hand gestures are gaining popularity as natural interfaces to interact with these devices. This work focuses on ego-hand gesture recognition in the context of interactions with virtual elements in AR and VR devices.

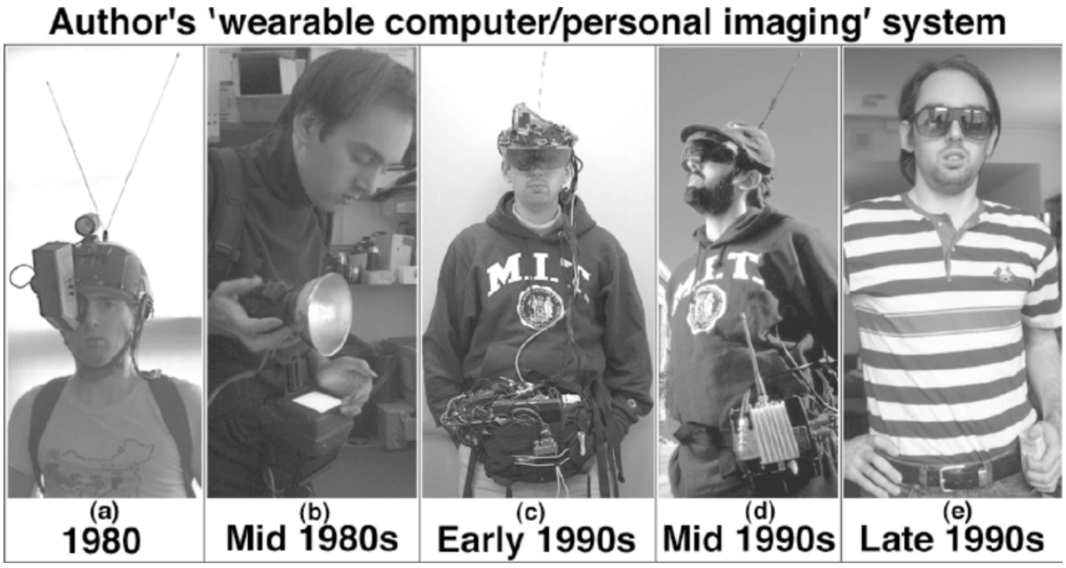


Figure 1.1: Evolution of WearComp, Prof. Steve Mann's wearable computing system.

1.1 Ego-hand Gestures

Presence of gestures is ubiquitous; they are found across cultures and ages. Gestures are movements of body parts, often accompanied by speech to convey some form of expression. They can be independent of speech, yet powerful enough to communicate. Though gestures can be performed with many body parts such as eyes, mouth, hands and any combination of these, for this document, we restrict gestures to those performed by hands. Ego-hand gestures are hand gestures performed by a person as observed by a wearable camera worn by the same person. Since gestures are an essential part of our daily communication, recognising gestures from an egocentric perspective and using them to interact with virtual elements is a natural extension from Reality to AR and VR.

1.2 Interactions in Augmented and Virtual Reality

Virtual Reality transports a person from the real world to a digitally constructed space that can stimulate the senses like the real world. Though the idea of virtual Reality has been existent before, the modern head-mounted VR system creation is primarily attributed to Ivan Sutherland and Bob Sproull in 1968. Since then, VR systems have been mainly used for medical, flight and automobile simulation [13, 14, 15].

Figure 1.2: Commercially available VR and AR headsets

Unlike VR, Augmented Reality (AR) aims to bridge the gap between virtual content and the real world. It tries to present information or content in a seamless and non-intrusive way to the user. See-through head-mounted devices (HMD) where the user is not isolated from the real world, and the virtual content displayed blends into the surroundings is one way AR achieves this objective. The term 'Augmented Reality' was coined by Thomas P Caudell, while he was working with his colleague David Mizell on creating an alternative to help factory floor workers understand wiring instructions for manufacturing processes. Though AR was initially intended for improving the productivity of factory workers and engineers when it was introduced [16], experimental applications of AR exploded to a multitude of fields like education, medicine, entertainment, gaming and tourism [17, 18, 19, 20]

The large scale production and adaption of both VR and AR systems into consumer space happened recently with the advancement of mobile computing, inertial sensing technologies, and computer vision algorithms. Commercially viable VR headsets like Oculus Rift, Samsung Gear VR, HTC Vive and AR headsets like Microsoft HoloLens, Magic Leap One (Figure 1.2) are gaining entry to into both enterprise and consumer space. However, natural interactions with virtual content is still a problem to be explored. Interactions form a majority of user experience; without easy usage, it is hard for technology to reach people. This is evident from the mobile phone revolution, one of the primary reasons for their pervasive presence apart from Moore's Law, is the fact that they are designed for everyone to understand their usage intuitively.

The current AR/VR headsets mostly use variations of joystick for facilitating interactions. Though some systems use hand gestures as a mode of interaction, their usage has been limited since ego-hand gesture recognition has not been extensively explored.

1.3 Ego-hand Gesture as Natural Interfaces for AR and VR

Gesture recognition, in general, is a hard problem to solve in computer vision. It involves many challenges like identifying hands of different skin tones, working under different lighting conditions, varying duration of the performance. Ego-hand gesture recognition adds an additional layer of complexity since it also involves ego-motion due to motion of device worn, partial or obstructed views of hands. Ego-hand gesture recognition should involve localisation in time and work in real-time while addressing all the problems mentioned above, to be useful for interacting with virtual content.

In recent times, deep neural networks have been discovered to provide robust solutions to address most of the problems mentioned earlier. With the motivation of recognising ego-hand gestures to design interfaces to AR and VR devices, the following **objectives** were envisioned for the research in this thesis.

- Define characteristics of a good ego-hand gesture dataset that can be used to train ego-hand gesture recognition deep neural networks. To identify the problems in existing publicly available ego-hand gesture datasets and provide a dataset that could potentially alleviate some of the identified issues.
- Design a deep neural network architecture that focuses on ego-hands in the images to improve the state of the art recognition benchmarks on existing datasets and the new dataset.
- Identify the problems in using the existing ego-hand gesture recognition networks, devise solutions to solve some of the identified problems to move towards more usable neural networks in real-world scenarios.

The thesis **contributions** are the following to realise the objective conceptualised above.

- Dataset of Ego Hand Gestures to address the problem of training new deep neural networks for ego gesture recognition with semi-automatic annotation(Chapter 4).
- A deep neural network architecture that creates embeddings pertaining to ego-hand and the gesture they are performing to the address the problem of recognising ego-hand gestures in trimmed videos(Chapter 5).
- An extension to Long-Short Term Memory(LSTM) framework to decrease dependency on heuristics, a new loss function and a novel evaluation metric to address the training of ego-hand gestures recognition in untrimmed videos which are more applicable in a real-world scenario(Chapter 6).
- In addition, the research done on ego-gesture recognition on untrimmed also lead to an on-going collaborative research project with NVidia ISAAC team. This work is explained in the Future Work section of the last chapter.

All these contributions lead to peer-reviewed and in-review publications listed below.

- Chalasani, T., Ondrej, J., & Smolic, A. (2018). Egocentric Gesture Recognition for Head-Mounted AR Devices. Adjunct Proceedings - 2018 IEEE International Symposium on Mixed and Augmented Reality, ISMAR-Adjunct 2018
- Chalasani, T., & Smolic, A. (2019). Simultaneous segmentation and recognition: Towards more accurate ego gesture recognition. Proceedings - 2019 International Conference on Computer Vision Workshop, ICCVW 2019
- Chalasani, T., & Smolic, A. (2021). EgoCentric Hand Gesture Recognition on Untrimmed Videos Using State Activation Gate LSTMs, **In-Review** IEEE Conference on Virtual Reality and 3D User Interfaces (VR)

Rest of the thesis is organised as follows: Chapter 2 introduces deep learning concepts pertaining to spatial, temporal, spatio-temporal learning and reason the necessity for large amounts of data needed to train these networks effectively. In Chapter 3, existing hand gesture recognition and ego-hand gesture recognition methods are reviewed, and publicly available ego-hand gesture datasets are introduced. Chapter 4 presents the Green Screen Ego-hand Gesture dataset, properties needed for an easy to use Ego-hand Gesture Dataset, and various novel attributes of the new dataset. Network architectures, training methods, loss functions, metrics created for trimmed and untrimmed ego gesture recognition are presented in Chapter 5 and Chapter 6. The final Chapter 7 concludes the thesis with a summary of the work done and possible future extensions.

Chapter 2

Concepts of Deep Learning

As stated earlier, the main contributions of this thesis are a new ego hand gesture dataset for training deep neural networks and new deep neural network architectures. It is imperative to understand the basic building blocks of deep neural networks for spatial, temporal and spatio-temporal learning and the necessity for large datasets. These concepts are discussed in the following sections.

2.1 Convolutional Neural Networks

The solutions to many computer vision problems like recognition, tracking, reconstruction, segmentation can be broadly classified into two categories, one using handcrafted features and other using auto extracted features depending on the data. Handcrafted feature descriptors like Scale Invariant Feature Transforms (SIFT) [22], Histogram of Oriented Gradients (HOG) [23], Speeded Up Robust Features (SURF) [24] were devised and used in ML frameworks like Support Vector Machines (SVMs), Artificial Neural Networks (ANNs) and others. Convolutional Neural Networks (CNNs) took this step away and let the data decide the features viable for the task in hand [25, 26, 27, 28]. These approaches had similar performance results in comparison to handcrafted feature until Krizhevsky et al. in [29]

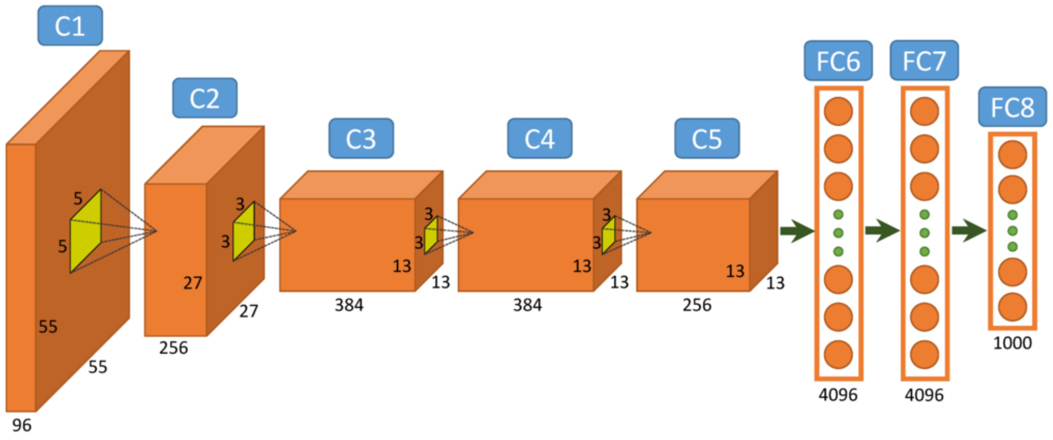


Figure 2.1: Network architecture of AlexNet. The figure here represents slightly altered AlexNet architecture, all the layers here were split into two for training on different GPUs in the original implementation [21].

came up with a Convolutional Neural Network (CNN) architecture that trained on a large set of images from ImageNet LSVRC (Large Scale Visual Recognition Challenge)-2010 [30] using specialised fast convolutional operations on Graphics Processing Units(GPUs). It should be noted that CNNs have been used for a long time before Krizhevsky et al. [29], like Jarret et al. [25], LeCun et al. [31], Lee et al. [26] to cite a few examples. Krizhevsky et al's critical contribution in [29] was optimised GPU implementation of 2D convolutions and other necessary operations for forward and backpropagation, finding the correct operations to deal with the inherent problem of overfitting(discussed in Section2.3).

Krizhevsky et al. [29] used ImageNet dataset [30] which contained 1.2 million high-resolution images with 1000 object classes of which 50% were used for training, and the remaining 50% were used for testing. They achieved a top-1 and top-5 error rates of 37.5% and 17.0% respectively doing considerably better than then state-of-art. The core practical principle of deep convolutional neural networks is a fast running implementation of convolutions on GPUs. The architecture proposed by [29] consisted of 5 convolutional layers, each appended with a ReLU to impart non-linearity and interspersed with Max Pooling layers. The fifth convolutional layer is then connected

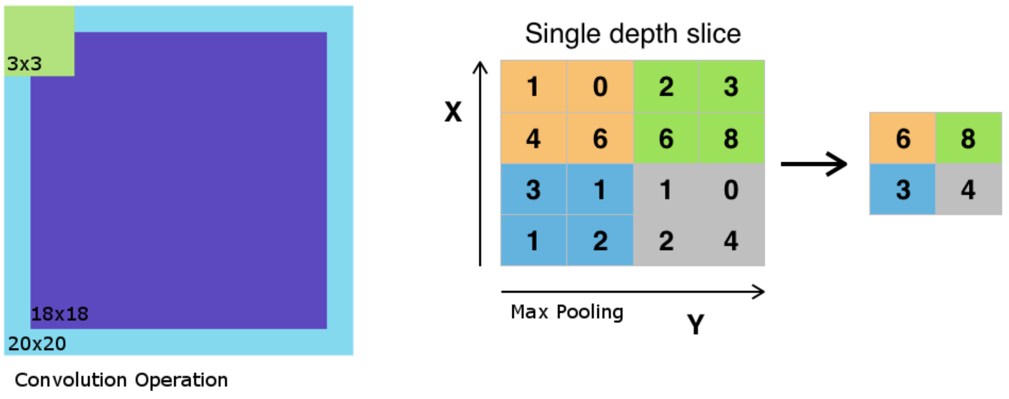


Figure 2.2: Figure to the left showing resulting sizes of convolution operation. The blue square is the input image, green square is the convolution window and purple one is the output. Figure to right showing max pooling operation [33].

to densely connected layers. The whole architecture (more famously known as AlexNet) is illustrated in *Figure 2.1*.

Convolutional Neural Networks when introduced mainly used 2D convolutions for extracting features from images, however research by Tran et al. [32] has lead to usage of 3D Convolutions for extracting spatio-temporal features from video data. In the following chapters, we look at various building blocks that make up 2D and 3D convolutional networks.

2.1.1 2D Convolutional Layer

A 2D Convolution Layer consists of N (usually referred to as depth) convolution filters of a specified size. Convolution is a simple mathematical operation which involves an input image (matrix) and a convolution filter (usually a small square matrix) to generate an output. If the input is an image I of size $N \times M$ (N-Rows and M-Columns), and the convolution filter is F of size w $\times$ w, the convolution between I and F produces an output image (matrix) of size $N - 2 \times \lfloor w/2 \rfloor \times M - 2 \times \lfloor w/2 \rfloor$ (Fig 2.2 showing sizes of input, window, outputs for example convolution operation). The values of output O can be calculated with standard convolution equation 2.1. It should be noted that this operation is highly parallel and ideal candidate to be implemented on GPU. As stated in the beginning, there are N such convolution filters defined in a convolution layer.

$$O_{i,j} = \sum_{a=1,b=1}^{a=w,b=w} I_{i+a-1,j+b-1} * F(a,b) \tag{2.1}$$

Each of these filters is applied to all the image channels and averaged to create one output per filter. There is an additional padding parameter that is specified. The padding adds additional rows and columns around the input image, usually used to make the input and output size the same. Stride is another parameter that defines how the filter window should move.

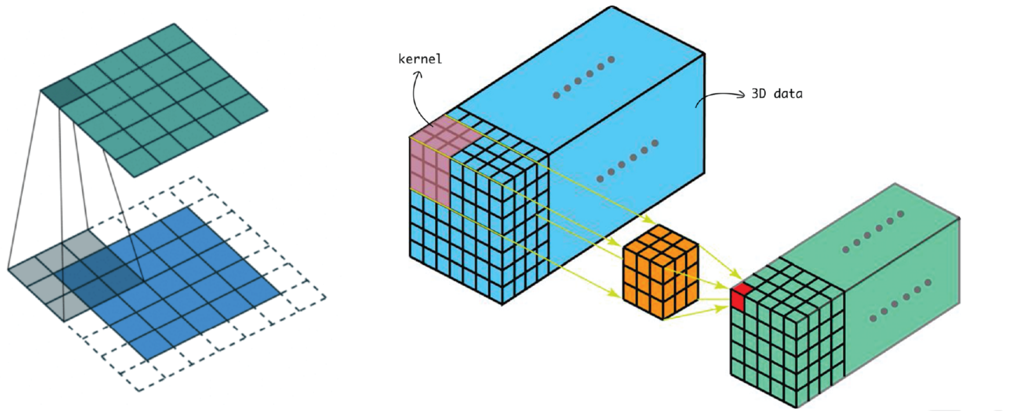


Figure 2.3: Figure showing 3D convolution [32] operation in comparison to 2D convolution. Figure sourced from [34].

2.1.2 3D Convolutional Layers

A 3D convolution filter is similar to 2D convolutions, however unlike the 2D case, 3D convolutions are applied to a sequence of images instead of a single image. The equation 2.2 for calculating the output of a 3D convolutional operation follows from the 2D equation 2.1. The convolutional filter in this case a 3 dimensional matrix F usually of a size $w \times w \times w$. If the image I of size $N \times M$ and there are L images in the sequence and is represented by I_L. Applying F

to I_L would result in a 3D matrix with size $(N - 2 \times \lfloor w/2 \rfloor) \times (M - 2 \times \lfloor w/2 \rfloor) \times (L - 2 \times \lfloor w/2 \rfloor)$

$$O_{i,j,k} = \sum_{a=1,b=1,c=1}^{a=w,b=w,c=w} I_{i+a-1,j+b-1,k+c-1} * F(a,b,c) \tag{2.2}$$

2.1.3 Activation Layers

Activation Layers are used to introduce non-linearities into deep neural networks.

ReLU

Rectified Linear Units are one of the activation functions used in Deep Neural Networks to introduce non linearity, it is defined by the equation 2.3. As explored by Nair & Hinton in [35] and Jarret et al in [25] compared to other activation functions ReLU provide faster convergence.

$$F(x) = max(0, x) \tag{2.3}$$

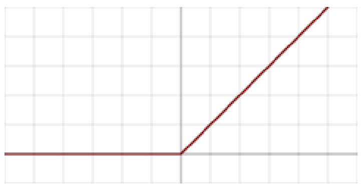
TanH

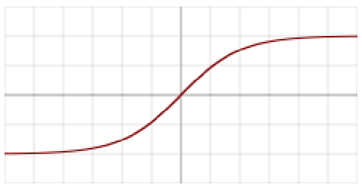
TanH function as defined by equation 2.4 is usually used in recurrent neural networks. The usage of this function will be explored later in section 2.2.2.

$$\frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{2.4}$$

Sigmoid

Sigmoid function in neural networks are used to map the output of a layer to $[0, 1]$. This function is defined by the equation 2.5. Fig 2.4 plots this function.


(a) ReLU


(b) TanH

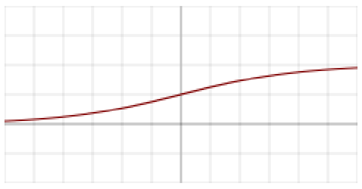

(c) Sigmoid

Figure 2.4: Various Activation functions. [33]

$$\frac{1}{1+e^{-x}} \tag{2.5}$$

SoftMax

Softmax activation or layer is used to map an input vector of J to probability distribution function containing J probabilities. This layer is usually used at the end of a full network for classification and is defined by the equation 2.6

$$\frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \tag{2.6}$$

2.1.4 Pooling Layers

Pooling layers are used for down sampling. There are several kinds of pooling layers but we will look at two kinds Max Pooling and Average Pooling.

Max Pooling Layer

This function looks at the maximum value present in the filter window and assigns it to the output. This windows can be overlapping or non-overlapping.

Average Pooling Layer

Average pooling calculates the mean of all values in the window and assigns it to the output. Just like Max Pooling layer, the windows can be overlapping or non-over lapping.

The building blocks of Deep Neural Networks explained dealt with extracting spatial features (Section 2.1.1) and spatio-temporal features in a windowed time frame (Section 2.1.2). However, these are not adequate to find long term temporal relations between data. In the next section, we look at a class of neural networks called recurrent neural networks(RNNs), that maintains a memory to extract long and short term temporal relations between data points in sequential/temporal data.

2.2 Recurrent Neural Networks

Recurrent Neural Networks were introduced to tackle the problem of sequence labelling, sequence to sequence mapping in speech and natural language processing settings initially [36, 37, 38, 39, 40]. The idea of using convolutions along the temporal dimension to understand the relationship between data points at different times has explored by a large body of research [41, 32, 42]. However, all these approaches suffer from the problem of not understanding long term dependencies, as they always work on a limited window of time. The underlying premise of recurrent neural networks model is the idea of estimating a hidden unit in current time based on its previous value and the current input to the network. This kind of estimation leads to the understanding of long term dependencies in a given sequence of arbitrary length theoretically. Subsection 2.2.1 delves a bit deeper into the essential flavour of RNN, its workings

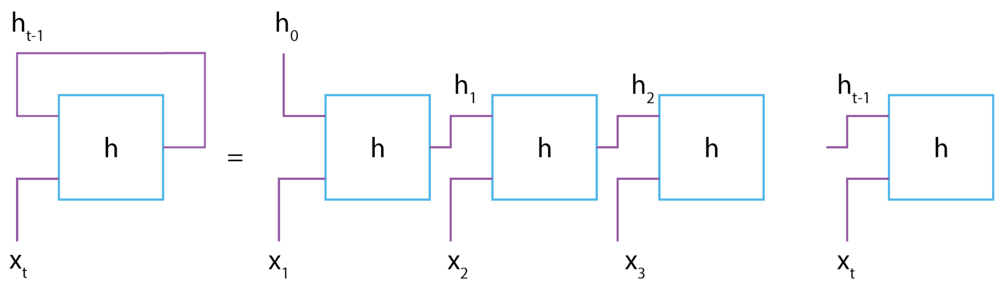


Figure 2.5: Figure showing a single RNN unit, and an unrolled RNN.

and pitfalls, and the next Subsection 2.2.2 introduces an improved version of RNNs that can tackle the problems of basic RNNs.

2.2.1 Basic RNNs

The idea of current state being a function of previous state is the basis of a dynamic system. Recurrent neural networks follow the idea of dynamic systems, and also consider the input from current time step to update the hidden state along with the hidden state from previous time step, as described by equation 2.7. They add a $\tanh$ activation described in Section 2.1.3 to add non linearity to the system. Once the hidden state is updated, the output of a recurrent unit is calculated as a function the updated hidden state. Equations 2.8, 2.9 describe a unit of RNN.

$$h^t = f(h^{t-1}, x^t; \theta) \tag{2.7}$$

$$h^t = \tanh(b_h + W_h * h^{t-1} + W_x * x^t) \tag{2.8}$$

$$o^t = b_o + W_o * h^t \tag{2.9}$$

Where x_t is the input, h_t is the hidden state and o_t is the output at time step t. W, b are the weights and biases. As it can be seen from figure 2.5 and the equations 2.7, 2.8, 2.9 the output of RNN at current time step is dependent on all the previous time steps. This concept is powerful enough to simulate a Turing Machine [43], however there are two practical problems of using RNNs which arise during training using back propagation:

- Exploding Gradient
- Vanishing Gradient

Backpropagation for calculating the weights in RNNs is done through loop unrolling. In this technique, as the weight updates from computing the gradients are applied, they either keep exponentially diminishing or increasing. The books Deep Learning [44] and Supervised Sequence Labelling with Recurrent Neural Networks [45] both give an excellent in-depth explanation of exploding and vanishing gradient problems faced while training an RNN. Hochreiter introduced Long Short-Term Memory (LSTM) and Schmidhuber [46] to specifically deal with this problem, the following Section 2.2.2 expounds on LSTM.

2.2.2 Long-Short Term Memory Units

The concept of Constant Error Carousal(CEC) which forms the basis of LSTM was introduced by Hochreiter and Schmidhuber [46] to deal with exploding and vanishing gradients. CEC was introduced as a mechanism to maintain the derivative of the recursive term to be approximately close to 1. In such a scenario, repeated multiplication over loop unrolling for long sequences would not lead to exponential increase or decrease in the gradient, giving a solution to the problems created during training a naive RNN.

Forget, input and output gates are used to maintain the flow of information over time to update the memory contents in LSTM, instead of direct updates done in a naive RNN (equation 2.8). The forget gate (equation 2.10) helps reduce the effects of irrelevant information from the previous state depending on the current input and hidden state.

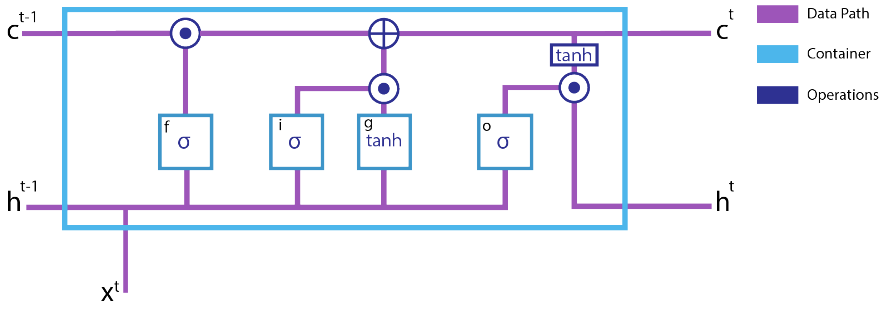


Figure 2.6: Forget, Input, Output gates of the LSTM that update the memory and hidden states

$$f_t = \sigma(W_{xf} * x^t + b_{xf} + W_{hf} * h^{t1} + b_{hf}) \tag{2.10}$$

The input gate (equation 2.11) selects the amount of relevant information that needs to be added to the previous memory state from the current input and hidden state (equation 2.12). This selection in combination with forget gate is used to update the current memory state(equation 2.13).

$$i_t = \sigma(W_{xi} * x^t + b_{xi} + W_{hi} * h^{t-1} + b_{hi}) \tag{2.11}$$

$$g_t = \tanh(W_{xg} * x^t + b_{xg} + W_{hg} * h^{t-1} + b_{hg}) \tag{2.12}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{2.13}$$

The final output gate (equation 2.14) regulates the amount of information from the current memory to update the hidden state (equation 2.15). The figure 2.6 shows the connections between the various gates of an LSTM and the flow of memory and hidden states inside it. The derivatives calculated for the hidden state and memory cell (h_t, c_t) in the cases above are not just directly dependent on their previous state, but through the gates that update them. This process helps them maintain CEC.

$$o_t = \sigma(W_{xo} * x^t + b_{xo} + W_{ho} * h^{t1} + b_{ho}) \tag{2.14}$$

$$h_t = o_t \odot \tanh(c_t) \tag{2.15}$$

2.3 Data

The first successful deep neural network with a practical workable implementation on GPUs was introduced by Krizhevsky et al. [29]. The total number of parameters that was estimated in the network is approximately 61 Million, table 2.1 summarises the parameters in each layer.

Layer	Size	Parameters to be Estimated
L_1 (Convolution)	$96 \times 11 \times 11 \times 3$	34848
L_2 (Convolution)	$256 \times 5 \times 5 \times 48$	307200
L_3 (Convolution)	$384 \times 3 \times 3 \times 256$	884736
L_4 (Convolution)	$384 \times 3 \times 3 \times 192$	663552
L_5 (Convolution)	$256 \times 3 \times 3 \times 192$	442368
L_6 (Linear)	9216×4096	37748736
L_7 (Linear)	4096×4096	16777216
L_8 (Linear)	4096×1000	4096000
Total		**60954656**

Table 2.1: Parameters in each AlexNet Layer

The input to this network is a 224×224 RGB image, implying the need for a large number of images not to overfit and estimate the 61 Million parameters to generalise well. This network was implemented on GPU with only 3GB of memory, with increasingly available GPU memory since then, the networks proposed grew larger requiring larger amounts of data to train. This requirement also gave rise to various techniques of data augmentation.

Krizhevsky et al. [29] employed label-preserving transforms as data augmentation techniques. They used random patch extractions, translation and horizontal reflections to inflate the training dataset by 2048 times the original size. Data augmentation techniques like these were inspired from earlier works by Simard et. al [47] and Ciregan et al. [48]. Without employee these techniques they observed a considerable amount of overfitting since the parameters to be estimated were way higher than constraints available.

Overfitting is the problem that occurs when a network learns the training dataset too well, losing it's capacity of generalise. In such a scenarios the training error and any performance metrics on the training dataset are optimised, but the test and validation dataset accuracies deteriorate considerably.

Krizhevsky et al. also used another form of data augmentation by varying the intensities of RGB channels of images in the training dataset. They performed PCA on the RGB pixel values from all the images in the ImageNet training set. They multiplied each eigen value with an random value drawn from Gaussian with mean 0 and standard deviation of 0.1, and multiplied with the corresponding eigen vector to reconstruct each RGB pixel in the training images. This kind of augmentation captured the property that identity of the images is invariant under changes in the intensity and colour of the illumination. This data augmentation technique reduced the top-1 error rate by 1%.

2.4 Summary

This chapter introduced the foundational concepts of deep convolutional and recurrent neural networks. We also looked at the necessity for large amounts of data that is needed to train these networks. In the following chapter, the existing research pertinent to gesture recognition and ego-hand gesture recognition before and after the advent of deep neural networks are looked into in detail. The publicly available datasets that can be used to train ego-hand gesture recognition networks are also reviewed.

Chapter 3

Literature Review

Prof. David McNeill in the book **Hand and Mind: What Gestures Reveal About Thought** [49] argued that hand gestures are as innate to human communication as speech itself. Before his work, most of the emphasis for human communication had been on verbal speech and gestures were considered an accompaniment to speech. However, David's work argued the contrary saying gestures of the hand are as integral to communication as verbal speech, went a step further to point to research that showed hand gestures as a communication tool preceded speech. Around the same time, seminal work by Tamura and Kawasaki [50] introduced research on gesture recognition through images to computer vision. They proposed a novel way to classify motion images(video clip) of deaf-and-mute sign language, under the supposition that the sign language word is a time sequence of units called cheremes. They described the chreme using hand-shape, movement, and hand location to make a dictionary for each word composed of cherems. They used this dictionary to recognise a sign language word from motion images.

This idea of using hand-shape, movement and location in different ways formed the basis of gesture recognition in both egocentric and non-ego-centric gesture recognition. Since non-egocentric gesture recognition paved the way for egocentric gesture recognition, it beckons us to understand the state of the art gesture recognition as a basis for egocentric hand gesture recognition. In the following section, we look at research related to non-ego-centric hand gesture recognition.

3.1 Gesture Recognition

Classic gesture recognition was different from the more recent approaches. Several handcrafted features SIFT [22], SURF [24] in conjunction with machine learning techniques like support vector machines, hidden markov models were employed. However, more recent approaches have seen a shift toward deep neural networks. Following this difference in approaches, gesture recognition before deep learning is discussed in the next section. The research based on deep learning is discussed in the one after.

3.1.1 Pre Deep Learning

One of the earliest attempts at gesture recognition was to interface with a robot arm, to move the robot arm using hand gestures [51]. Torige and Kono achieved this by using a coloured glove that is different from the surroundings and extracting the coloured area giving the approximate location of the fingers. They used a stereographic camera to record stereo images. The 3D location of fingers is calculated per frame using standard procedure since the point correspondences are available from the calibrated stereo rig. The 3D positions from the image sequence are used to classify the gesture and interface with the robotic arm to move four directions. Though a simple system by current standards, it was a precursor to complex systems developed since then.

Darrell and Pentland eliminated the use of coloured gloves, by tracking the template of a hand with Normalised Cross-Correlation (NCC) across a give sequence in [52]. The algorithm then uses the NCC score of each image in the sequence as time-series information and predicts the gesture performed using Dynamic Time Warping algorithm. However, this approach requires a model to be generated per gesture per user to achieve good recognition accuracy.

Davis and Shah [53] proposed a method to use markers on fingertips(akin to a coloured glove), track the trajectories of markers as vectors. They proposed dividing the performance of gesture into four phases and used a finite state machine to track the changes from one phase to another. The trajectories of each fingertip are calculated according to motion correspondences calculated using [54]; each gesture is then modelled and encoded into motion bit vectors using the magnitude and direction of the trajectories. The gesture recognition is done by matching the encoded bit vectors to those stored in the dictionary.

Hidden Markov Models (HMMs) were introduced to gesture recognition as sequence classification technique in [55] by Starner et al. taking inspiration from [56] for using HMMs to classify a sequence of images. They create handcrafted feature vector of size eight containing each hand's x and y position, angle of the axis of least inertia, and eccentricity of bounding ellipse per each image. This feature vector is calculated per image. The sequence of feature vectors is used with embedded training strategy to train an HMM for sequence classification. It should be noted that the method still needs the use of a coloured glove to identify the hand in an image. Min et al. [57] used HMM-based framework to recognise hand gesture to interface with a graphics software. It was one of the first attempts to use hand gestures as an interface to interact with virtual environments.

Segen et al. introduced Finite State Machine (FSM) without the use of coloured markers for gesture classifications in [58]. Similar to [51] they built a stereo-rig and used image subtraction for the hand extraction process, which makes this method not capable of handling dynamic backgrounds. Once the hand is extracted from each of the two images from the stereo rig, they calculated 2D correspondences. These correspondences are then used to generate 3D pose of the hand. They looked at the curvature along the boundaries of the hands, defined peak as the feature with positive curvature and valley as the feature with negative curvature at the local extremas. These peaks and valleys are used to do a coarse recognition of gestures into four predefined classes. Then the features are again used in a finite state machine to perform classification at finer levels. It is one of the first methods to forgo the use of coloured gloves and have a unified model for all gestures independent of the users.

The previous algorithm in [58] is extended by Du et al. in [59] by using skin detection filter developed by Zarit et al. in [60] and excluding the use of a stereo-rig. Their system follows similar architecture to [58], replaces the 3D pose generation module with skin detection module to perform segmentation. Once segmentation of hand is done, peak and valley features defined in [58] are used in FSM architecture to recognise hand gestures.

Yoon et al. in [61] built a system from the ground up. For hand segmentation, they use YIQ colour space to reduce light intensity and build a histogram-based model for hand colour detection. Their system has an initialisation step where the user has to show the hand for a few seconds to build up the colour histogram model. They combine 4x4 pixels into a cell and perform histogram intersection algorithm proposed by [62], to classify each cell as a region belonging to hand or not based on a variable threshold. They employed spotting rule to isolate an identified gesture, meaning when the user wants his/her gesture to be recognised by the system, the hand needs to be static for few seconds before and after the gesture is performed. Once a sequence of images is identified to be belonging to a gesture, three different features (location, orientation, velocity) are extracted per image, and a k-means clustering algorithm is used to generate discretised features for HMM codebook.

Ramamoorthy et al. introduced a hand tracking component using Kalman Filtering techniques in [63], which gave the ability to track hand under cluttered background, variable lighting condition and reduced the necessity to detect hand in every frame. A hand in a frame is detected using probabilistic contour discriminant, in this method the contours of a hand pose models are stored as reference for the class of hand shapes. The user is expected to place the hand in the centre of the camera, the contour of hand is extracted using colour segmentation and then matched to one of the templates stored using contour discriminant proposed in [64]. Once there is a close match, the contour region is sent for tracking, the output of tracker is sampled periodically, and contour-discriminant-based features are extracted which are used as inputs to HMMs for recognising gestures.

One of the more recent works by Suk et al. in [65] used Dynamic Bayesian Networks (DBNs) to model gestures. DBNs are a more generalised version of Hidden Markov Models in the sense that they can have multiple independent states at a give time instance. In contrast, HMMs are limited to one hidden state variable. DBNs being more expressive than HMMs can theoretically represent gestures which are a complex series of events in time better. Face and hands in the images are detected separately using HSV colour histogram with Haar-detectors and colour range model in YIQ colour space correspondingly. Once a hand is detected, a two-dimension Gaussian is fit around the hand, and optical flow is used to track it. Features like location, the relative position of the two hands, the position of each hand with respect to the face are extracted. The proposed DBN architecture has three hidden states two states to represent left and right hands and the third state to resolve ambiguities between similar gestures. The observed inputs location and distance from the face are feed to the corresponding first two hidden states. The fifth observed input, which is the relative position of two hands is feed into the third hidden state. One DBN is created per gesture, and the whole network is trained cyclically with the inclusion of a network for filler gestures. This procedure resulted in a network that could identify continuous non-isolated gestures improving over the previous methods.

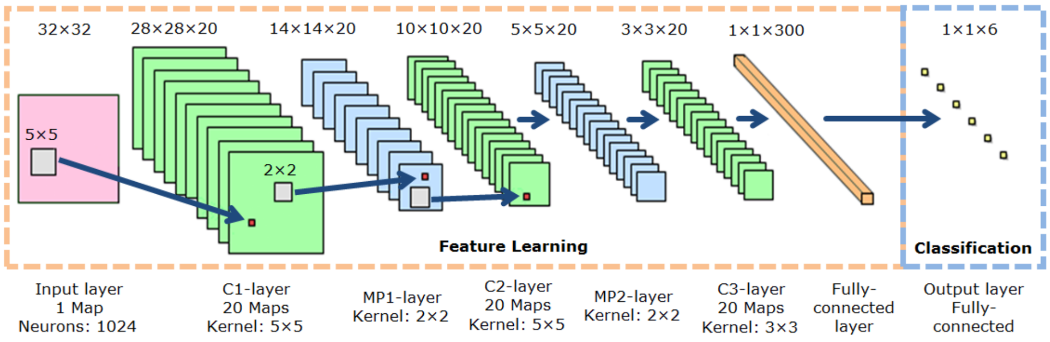


Figure 3.1: MPCNN architecture using alternating convolutional and max-pooling layers [66].

3.1.2 Deep Learning based Hand Gesture Recognition

Nagi et al. were first to apply deep learning to gesture recognition problem in [66]. Their proposed approach used a coloured glove to segment hand. The YCbCr colour space containing luminance (Y) and chrominance (CbCr) is used to deal with uneven illumination. The glove colour chrominance is modelled as an elliptical Gaussian joint Probability Distribution Function. They used Mahalanobis distance, thresholded to a practical value to categorise a given pixel as belonging to the glove or not. The binarised image of the segmented hand is preprocessed with smoothing filter, resized and padded to 32×32 pixels and used as an input to a Deep Neural Network. They defined six static gestures, created a dataset with 6000 images with 60/40 split for training and testing. Their network architecture similar to AlexNet is illustrated in *Figure 3.1*. Compared to methods that involved creating handcrafted features like FFT, Spatial Pyramid, PHOG coupled with SVM architecture performed well. It has to be noted that this method still needs the user to wear a coloured glove, and there is also a need to segment the hand separately before classification. Though it has limitations compared to methods we have described until now and not end to end learnable, it still is one of the first paper to apply Deep Learning to solve the problem of gesture recognition.

One of the first end-to-end learnable gesture recognition system using CNNs was designed by Barros et al. in [67]. They used a Multi

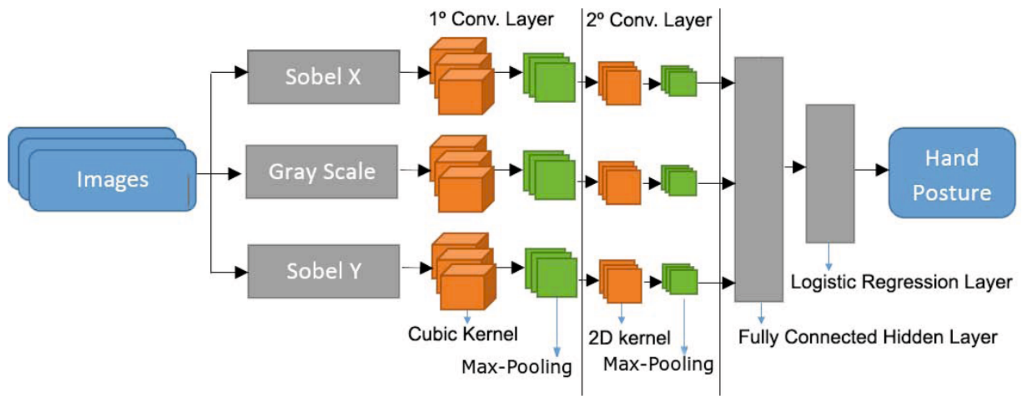


Figure 3.2: Multi Channel CNN architecture proposed in [67] for gesture recognition.

Channel CNN (MCCNN) architecture to achieve this. They also applied a cubic filter concept in their first layer, unlike a 2D convolution filter a cubic filter operates on a stack of images moving in all three directions. For their training, they stack together images which belong to the same gesture class as input to their MCCNN, and for the testing purpose, they use the same image multiple times as input. The hypothesis is that the cubic kernel can learn variance in gestures belonging to the same classes quicker. They also calculate Sobel images in both X and Y directions and supply them with the original greyscale image as three-column input to the MCCNN. The three columns in their architecture share a similar structure for layer one and layer 2, but the weights are independent. The outputs of all the columns after layer two are connected through densely connected layers, and this densely connected layer is connected to a Logistic Regression layer. *Figure 3.2* illustrates the MCCNN architecture used. They used the database defined in [68], which has ten hand postures, executed by 24 people with three different backgrounds to validate their model.

Ji et al. introduced 3D/cubic kernels to infer motion information from a sequence of images and perform human action recognition in videos in [69]. Unlike [67] they use the sequence of images from videos as input to the 3D Convolutional Neural Networks for simultaneously extracting spatial and temporal features. Tang et

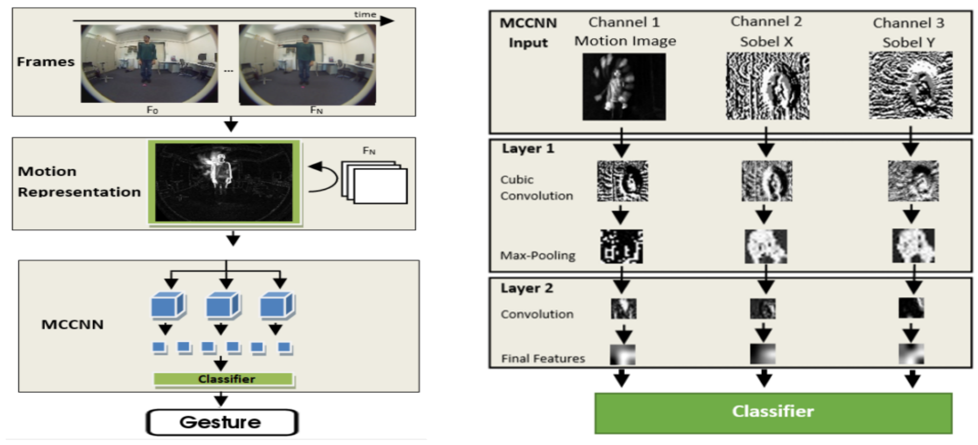


Figure 3.3: Multi Gesture recognition framework from [70], the architecture of Deep Network is similar to the one from *Figure 3.2*, but the input includes motion images to represent temporal dimension. This addition gives the network ability to detect dynamic gestures.

al. in [70] exploited this idea of using 3D cubic kernels for creating spatio-temporal features and combined this strategy with MCCNN proposed in [67] and came up with an end-to-end learnable network that can identify dynamic gestures. The architecture is similar to the one illustrated in *Figure 3.2*, the only difference here is the input. Input to the network in this architecture is a sequence of images where the subject performs the gestures. *Figure 3.3* illustrates the architecture, you can see that it is the same as the one from [67] but the only difference being the input to one of the columns, where the standard greyscale image is replaced by motion images (similar to optical flow). This addition of motion images gives their system the ability to recognise dynamic gestures since their network can infer temporal features from this input column.

The ChaLearn Dataset and challenge was introduced in 2012 for action and gesture recognition. The improved version of this dataset published in 2014 was one of the first data sets big enough to be usable for Deep Learning [71]. It had multimodal data containing the depth and intensity of each hand separately, and full-body pose information all synchronised together. Paper [72] used all the modalities together in a deep learning framework and achieved top per-

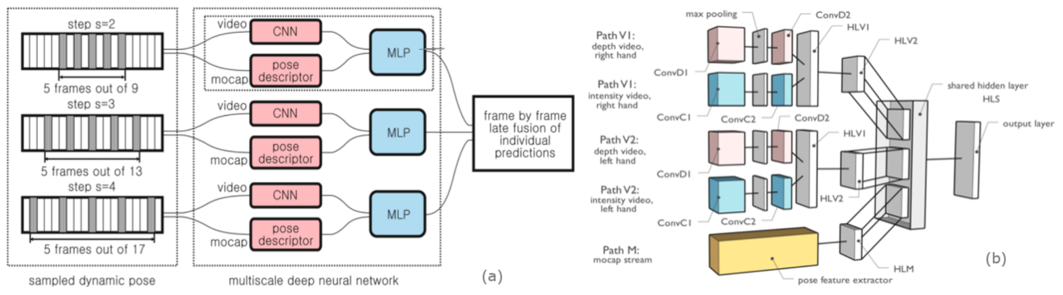


Figure 3.4: Fig (a) shows the multi-scale framework from [72] that deals with different duration of gestures. Fig (b) illustrates the network architecture, the same architecture is used thrice in the multi-scale framework.

formance in the competition. It was also one of the first works to use multimodal information successfully in deep learning applied to gesture recognition.

They also introduced a framework that could handle gestures with varied duration, which they termed as multi-scale. As illustrated in *Figure 3.4*, the input sequence is sampled at three levels and sent to the three networks independently. All the networks shared the same architecture, which contained three modules to deal with right-hand data (greyscale and depth), left-hand data(greyscale and depth) and pose data separately. They train the three modules separately in the first phase and do a late fusion with densely connected layers in the end with weights transferred from the first phase (see *Figure 3.4*).

R3DCNNs, a combination of recurrent neural networks (RNN) and 3DCNNs to recognise hand gestures, was introduced by Molchanov et al. in [73]. Similar to [72] they use three modalities depth, RGB and infra-red. Unlike CNNs, RNNs can model a sequence of data. Thus they have been used heavily in speech processing and natural language processing. They have been recently introduced to computer vision tasks like video recognition, activity recognition, image caption generation [74, 75, 76]. Molchanov et al. use CNNs generated output akin to a small encoding as input to RNNs to get frame by frame label given a sequence of images. They have published a dataset collected with a set up of RGB, depth and infra-red cameras, consisting of 25 different gestures performed by 20 different subjects. The dataset consists of a fair amount of data for training deep learning models, and their system set up is illustrated in *Figure 3.7*. They train the R3DCNN network (architecture illustrated in *Figure 3.7*) with each modality separately and instead of using a fusion layer like [72], they average the class-conditional probability vectors of each of the three models and supply it as an input to the RNN in the final step. They train the model in two phases; the first phase involves the training of the three models with different modalities independently. Furthermore, in the second phase, they average the output of three models, supply it as an input to the RNNs and train the entire R3DCNN together. With this training strategy and R3DCNN model no only did they achieved better results on the dataset they published compared to other state of the art methods, but also they perform well on ChaLearn2014, and SKIG RGBD datasets [71, 77]

Gesture recognition should work on untrimmed videos for being useful in application scenarios. Untrimmed videos consist of gestures performed by a person interspersed with frames without gestures. In most of the cases above, all the networks assume one gesture per video which is not the case in the real world. Gesture recognition from an untrimmed video has been dealt with a combination of Gesture spotting, and gesture recognition [78, 79, 80]. Gesture spotting identifies the beginning and ending from an untrimmed video once the gesture is made and the frames between the beginning and end are sent to a separate algorithm or network for recognition.

Lee et al. [78] solved the gesture spotting problem under the assumption that when a gesture begins and ends with the person's hands near the legs in standing position under a third-person view as illustrated in Figure 3.5. Once the frames with gestures are identified, Motion Histogram Images(MHI) [81] are calculated, and then Histogram of Gradient features are calculated on these MHIs to identify the gesture. Similarities between the gesture boundaries and recognition after the completion of the gesture were the limitations

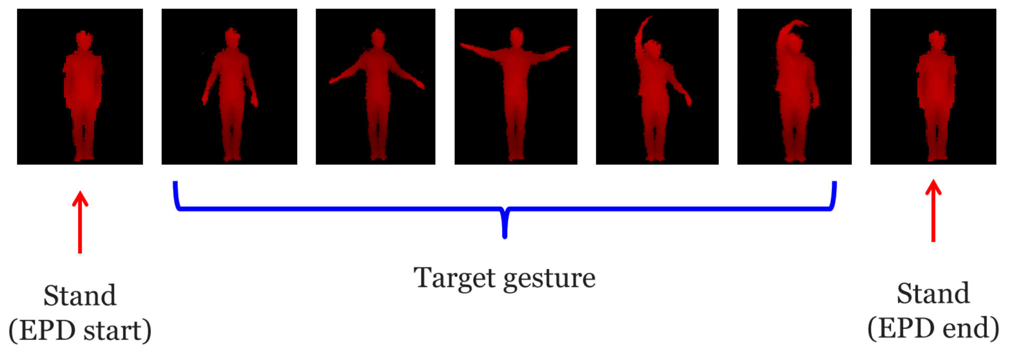


Figure 3.5: The beginning and ending of a gesture is identified by recognising the pose of the person if the hands are near the legs of the person. This strategy was introduced by Lee et al. in [78]

of Lee et al.'s work.

An end-to-end gesture segmentation/spotting network, as shown in Figure 3.6 with Temporal dilation and a balanced square hinge loss was proposed by Zhu et al. [79]. Once the gesture boundaries are identified, they proposed another network with 3DCNN + ConvLSTM + 2DCNN to recognise the gesture between the boundary frames. This method does not impose the restriction of the boundary frames to have similar pose as the previous one [78] however, the segmentation and recognition are again disjoint, and recognition happens after the gesture is completed.

Benitez-Garcia et al. [80] proposed a conceptually similar method to [78] to spot and recognise gestures in a car to interact devices in the car in a touch-less manner. Their spotting algorithm was based under the assumption that the beginning and ending boundaries of gesture frames have similar finger poses in their particular scenario. They experimented with handcrafted features (Histogram of Gradients and Histogram of Optical Flow) and deep learned features, defined the boundaries based on the similarity score of these features. Once the boundaries are detected similar to [78, 79], they used a different framework to recognise the gesture.

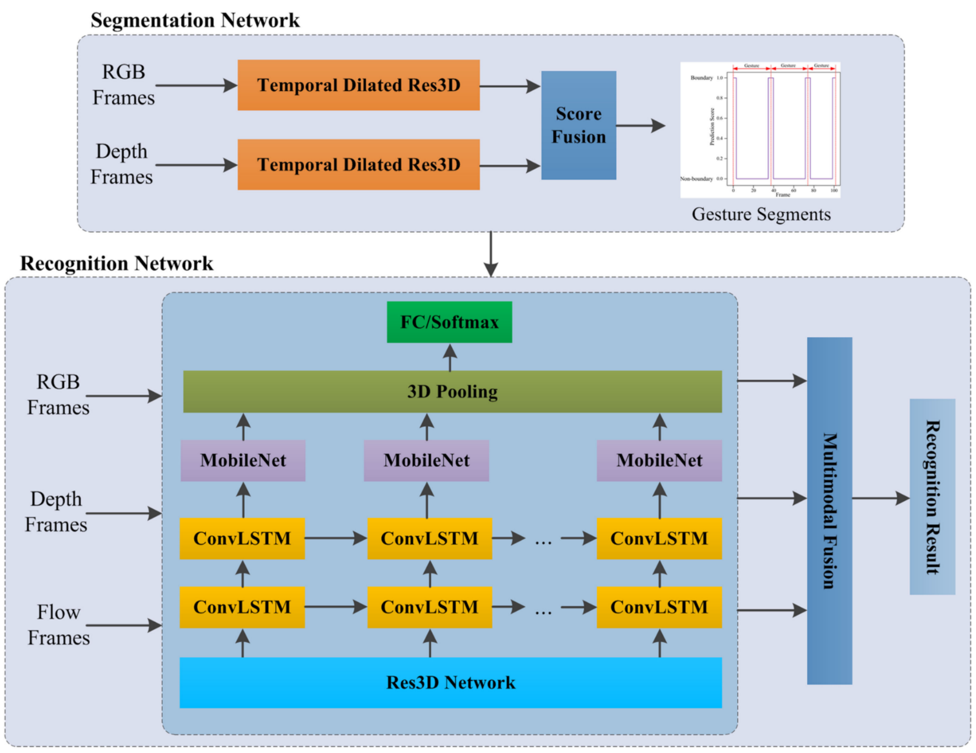


Figure 3.6: Zhu et al. [79] proposed a two different networks one for isolating gesture frames from continuous video with Temporal Dialated Res3D modules and another with 3DCNN + ConvLSTM + 2DCNN for recognising the gesture from isolated gesture frames.

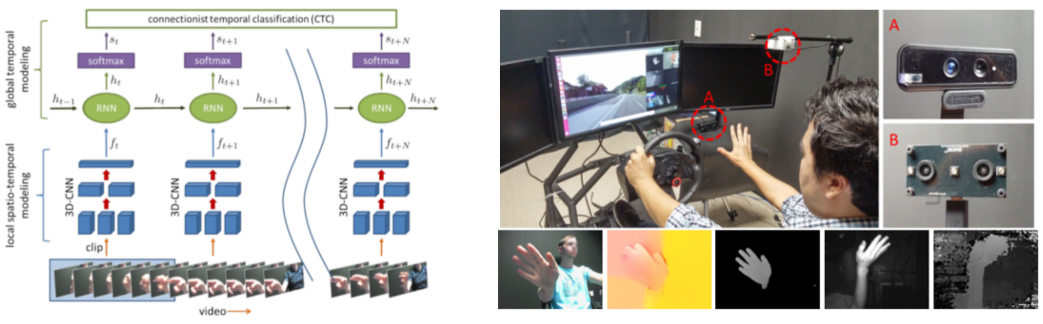

connectionist temporal classification (CTC)
global temporal modeling
local spatio-temporal modeling
softmax
RNN
3D-CNN
clip
video
A
B

Figure 3.7: Figure to the left shows the architecture of R3DCNN used in [73] for recognising gestures from different modalities. Figure to the right shows the system used to collect RGB and Depth data.

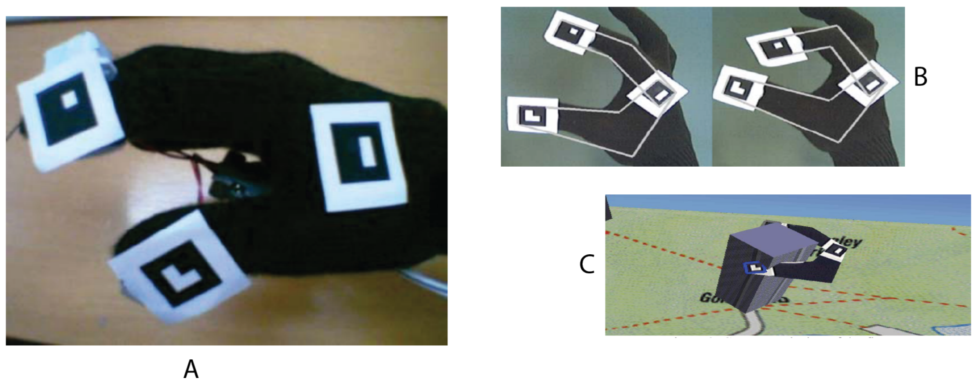


Figure 3.8: One of the first interactive AR system based on gesture recognition introduced by Buchmann et al.[82] A: They placed three markers on the tip of the index finger, thumb, and the joint between the two. B: The pose of the hand is calculated by continuously tracking the three templates relative positions. C: Their system worked under partial occlusions.

3.2 Ego-hand Gesture Recognition

Similar to gesture recognition the landscape of research changed after the advent of deep learning. So we look at research before and after deep learning in the following subsections.

3.2.1 Pre Deep Learning

Buchmann et al. [82] engineered a system called FingARTips to interact with virtual elements in AR in 2004. They used ARToolKit [83] as the main framework to track templates attached to the thumb, index and middle fingers of the user as illustrated in Figure 3.8A. The gesture is recognised by tracking the positions of unique markers placed on the three fingertips. Once the markers are tracked, their relative positions to each other are calculated, as shown in Figure 3.8B. As shown in Figure 3.8C, their system could handle gestures when interacting with the virtual object under partial obstruction as long as at least two markers were in view.

Li and Kitani [84] were one of the first to have proposed identifying hands from egocentric videos without using markers or gloves. Similar to standard gesture recognition identifying the hand was

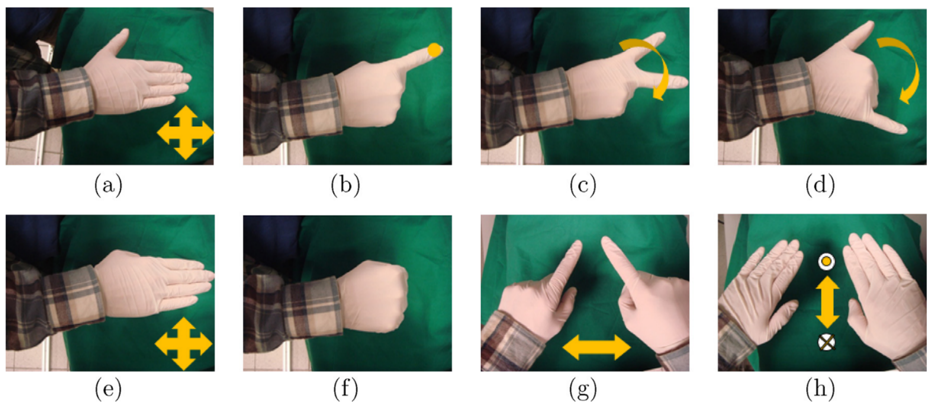


Figure 3.9: Dynamic hand gestures for robot and AR control. (a) AR – move; (b) AR – pointing; (c), (d) robot – arm rotation; (e) robot – x/y/z axial motion; (f) robot – stop; (g) AR – zoom in/out; (b) robot – suspend [86].

the first step used in ego gesture recognition. However, the set of problems for ego-hand recognition consisted considerable variation of backgrounds due to ego-motion, the variation of lighting conditions because of change from indoor to outdoor environments, motion blur introduced due to constant movement of the device wearer, which were not inherent to third-person view, static camera hand/gesture recognition. Li and Kitani proposed a combination of local appearance based colour and texture features of the hand and a global illumination dependent colour histogram using t-SNE [85] to achieve pixel-wise hand detection in egocentric views.

Wen et al. [86] proposed using gestures as an interface in AR assisted robotic surgery. Their work proposed using Kinect placed over the top of the surgery table (as shown in Figure 3.10), they identify the hands with depth data from Kinect. They defined eight different gestures that were used to control the assistive robot Figure 3.9. Inspired by [55] they trained HMMs to recognise hand gestures from handcrafted features like hand motion trajectory.

In [10] Serra et al. approached ego-hand gesture recognition in a similar manner to non-ego-hand gesture recognition. They proposed segmenting the hand using Simple Linear Iterative Clustering (SLIC) algorithm to extract superpixels. Besides, they used the

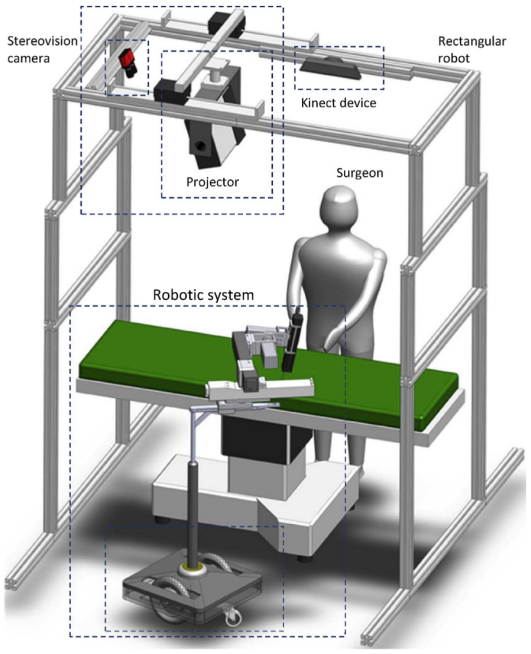


Fig. 2 – Overview of the cooperative surgical robot system.

Figure 3.10: Cooperative Robotics System: Kinect is placed on the top to capture hand movement which was used for recognising gestures for AR system [86].

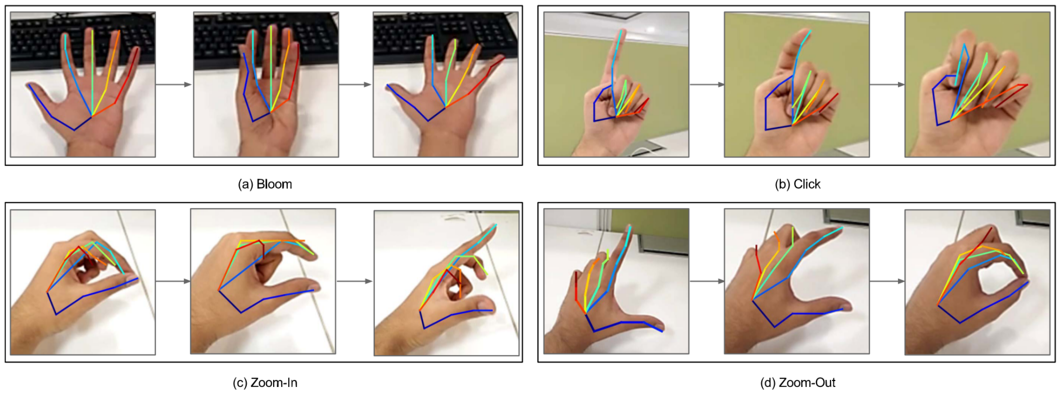


Figure 3.11: 21 key points recognised using HandPoseNet from [87] illustrated on 4 hand gestures that are recognised in [88].

previous frame's segmentation to introduce temporal consistency and GrabCut algorithm to introduce spatial consistency. Once a segment of hand is detected, they used Exemplar-SVM [89] to classify the segment into one of the five defined gestures. This method does not deal with dynamic gestures.

Baraldi et al. introduced the usage of dense feature trajectories to identify both static and dynamic gestures in [90]. They used hand segmentation method proposed in [10], to extract regions of hand and feature points are sampled from the segmented hand area and tracked during the gesture. Spatio-Temporal feature descriptors like Histogram of Gradients, Histogram of Optical Flow, Motion Boundary Histograms are calculated. Using these feature as Bag of Words an SVM is trained to classify the gestures. It should be noted that homography is calculated between each frame. The head motion is compensated with warping the current frame to the earlier frame using the homography. Baraldi et al.'s work was one of the first attempts to consider head motion compensation in ego-hand gesture recognition.

Hegde et al. were one of the first to introduce ego-hand gesture recognition algorithm that could run on a mobile device in [91]. The hand is segmented using the chroma channels C_b, C_r in YC_bC_r colour space using the Gaussian Mixture Model. Once a hand is detected feature points are extracted and tracked using the Lukas-Kanade flow tracker. They only recognise two gestures, swipe up and swipe down, which they distinguish by using the direction of flow per frame and giving the maximum label for the entire sequence.

We now look at Ego Centric Gesture recognition using Deep Learning algorithms.

3.2.2 Deep Learning based Ego-hand Gesture Recognition

As we have seen earlier, egocentric and non-egocentric gesture recognition both need to operable on untrimmed videos for the algo-

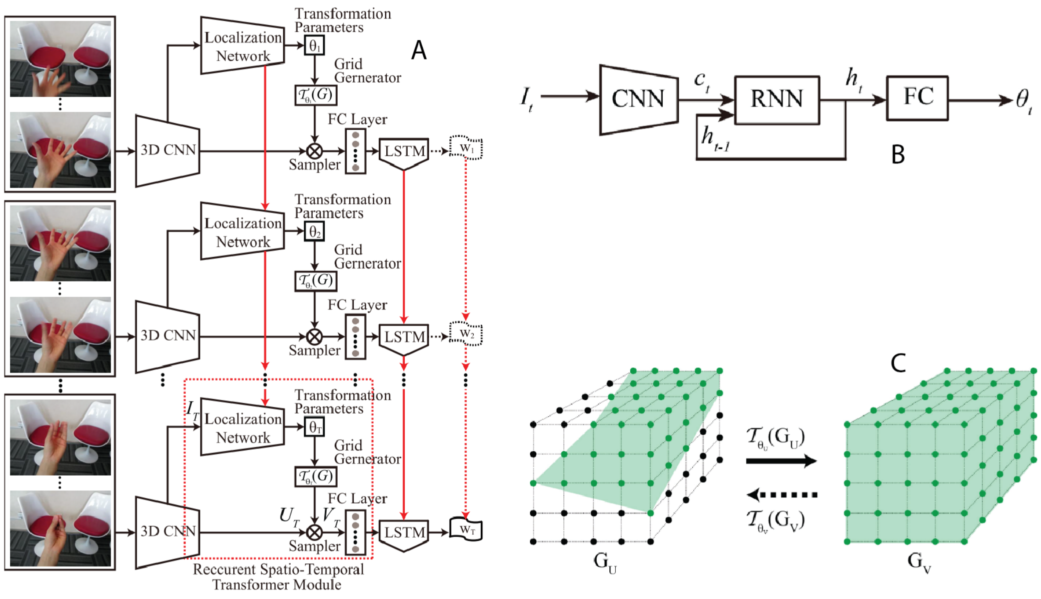


Figure 3.12: A: Full network architecture proposed by Cao et al.[12]. B: Structure of the Recurrent Spatio-Temporal Transformer Module(STTM). STTM module recurrently proposed an affine/homography transform which was applied to 3D features created by the prior network. C: Illustration of the grid before and after the transformation generated by STTM is applied to the grid features.

rithms to be practically applicable. However, exploring deep neural networks that recognise egocentric hand gestures on trimmed videos is worth studying because some of the methods from these can be transferred to recognition on untrimmed videos. We look at some of the recent network architectures that deal with egocentric hand gesture recognition on trimmed videos followed by research performed on untrimmed video ego-hand gesture recognition.

Ego-hand Gesture Recognition from Trimmed Videos

The method proposed in [88] by Jain et al. used per frame pose information of hands generated as described in [87] using HandPoseNet and passes this information to an LSTM(Section 2.2.2) network to identify four dynamic gestures. HandPoseNet is a Deep Neural Network architecture that is composed of modules that segment hand and input the segmented hand mask to a PoseNet that identifies 21 key points on a hand (Figure 3.11 illustrates the 21 key points on a hand while performing the four gestures). We briefly described Recurrent Neural Networks in *Chapter 2.2.1* as networks that could recognise time sequence information. However, they often suffer from the problems of exploding gradients or oscillating weights. LSTM networks were introduced by Hochreiter et al. in [46] particularly to deal with these issues. The method proposed by Jain et al. using the 21 key points generated by HandPoseNet per frame as time sequence input to an LSTM(described in *Chapter 2.2.2*) to recognise the gesture.

Cao et al. proposed the first end to end learnable architecture that can recognise ego gestures in [12]. They incorporated a recurrent Spatio Temporal Transformer Module that automatically finds the corrective homography or affine transformation and applies it to the feature maps generated by a 3D CNN. These transformed feature maps are then sent to LSTM for recognising the gesture. *Figure 3.12A* describes the network architecture proposed; it has four main components. A 3D CNN to extract spatial and temporal features, a localisation network to estimate the transformation from the current to the previous frame, a grid sampler that applies the estimated transformation to feature maps generated by the 3D CNN and finally an LSTM that take these transformed feature maps as inputs to recognise the gesture.

In [92] Abavisani et al. proposed a method to learn from different available modalities. They posited that under the availability of different modalities like RGB, Flow, Depth, each modality brought in different strengths and proposed a framework in which network trained on one modality can optimise their weights better if the performance of network learned on another modality is better. To achieve they used proposed a Spatio-Temporal Semantic Alignment loss(SSA Loss), summarised by the Equation 3.1

$$l_{ssa}^{m,n} = \rho^{m,n} \parallel corr(F_m) - corr(F_n) \parallel_F^2 \tag{3.1}$$

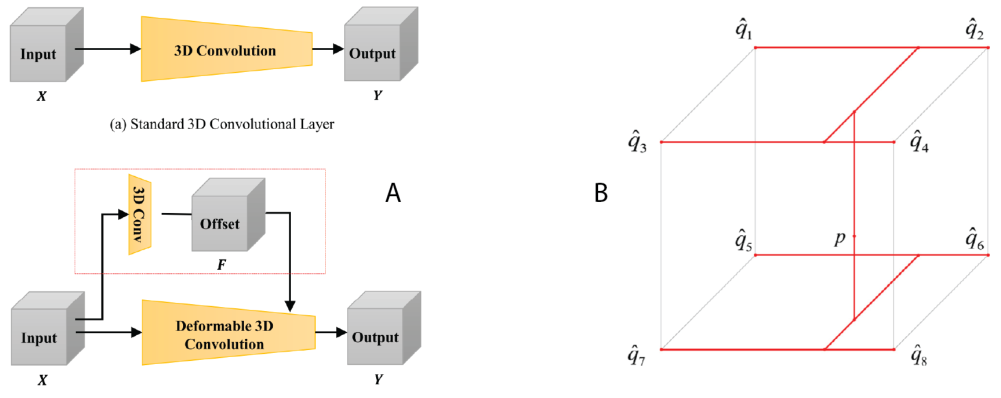


Figure 3.13: A: 3D Deformable convolutions proposed by Zhang et al.[94]

Where $corr(F)$ denotes covariance of matrix F. The $SSALoss$ is based on covariance matrix alignment from source to target. The underlying assumption for $ssaloss$ is that networks trained on modalities m and n should develop correlated spatiotemporal blocks extracting similar semantic features from inputs. In this case, if modality n has better performance over modality m, the loss dynamically regularised by ρ pushes the covariance of F_m closer to that of F_n. Using this strategy and I3D [93] as backbone they achieved better performance than RSTTM proposed by [12] on EgoGesture dataset.

Zhang et al. [94] proposed deformable 3D convolutions to deal with selectively applying spatiotemporal convolutions. They argued that while Cao et al. [12] calculated homography/affine transformations and applied them to the entire grid, applying spatio-temporal convolutional selectively through their deformable 3D CNN modules yields better focus on hands and their movements which lead to improved results on the EgoGesture dataset. Their method involved learning $3 \times 3 \times 3$ kernels which provided offsets to normal 3D convolutions (illustrated in Figure 3.13A). Equation 3.2 represents the standard 3D convolution.

$$Y(p_o) = \sum_{p_n} W(p_n).X(p_i + p_n) \tag{3.2}$$

Where p_i is the input point, p_n are the points around p_i. The Deformable 3D Convolutions are represented by Equation3.3.

$$Y(p_o) = \sum_{p_n} W(p_n).X(p_i + p_n + \delta_{i,n}) \tag{3.3}$$

The $\delta_{i,n}$ is dependent on the input and the neighbouring pixel selection, giving the kernel a deformable nature. The offset learned was fractional, to get a differentiable network they applied tri-linear interpolation as illustrated in Figure 3.13B to generate the final output. With 3D Deformable Convolutions added to the later part of network yielded good results for recognition on EgoGesture dataset.

We saw the research done on Ego Gesture recognition on trimmed video so far, in the next section, we explore the research available on ego gesture recognition on untrimmed videos.

Ego-hand Gesture Recognition from Untrimmed Videos

Untrimmed videos contain gestures being performed interspersed with a lot of inactivities or other activity. These two phases in the video must be identified to recognise the gestures in the video. As we have seen in Section 3.1 gesture isolation/spotting and then recognising the gestures is one way to deal with untrimmed videos. However, this technique limits the interface designers in AR/VR environments since it inevitably introduces lag between the performance of the gesture and the response from the system. There has been some work done in action recognition on untrimmed videos [95, 96], which are reviewed below, then the two available works [9, 97] on untrimmed video ego gesture recognition are reviewed.

Buch et al. [95] proposed the use of semantically constrained recurrent modules to deal with action detection, which is akin to gesture spotting. The features from input video are encoded using C3D network, and the encoded features are sent to the Recurrent module for further processing. The semantically constrained recurrent modules consisted of a n layers of Gated Recurrent Units (a kind of RNNs), and the output from the final hidden layer at each temporal step t is divided into K segments which are evaluated fur-

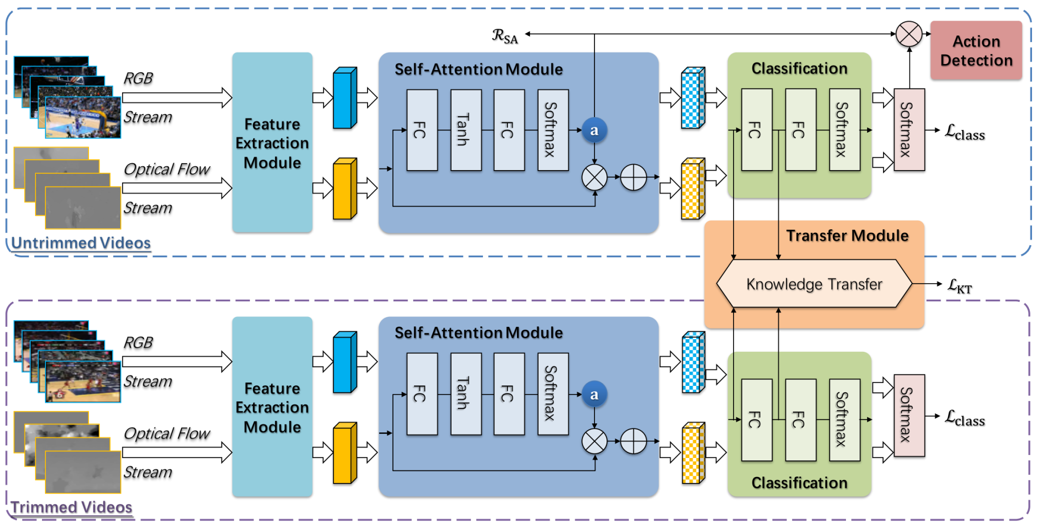


Figure 3.14: Two networks with the same architecture, one trained on untrimmed video and another trained on trimmed video. The weights are transferred using the transfer module from trimmed to the untrimmed network to make its recognition better [96]

ther to check for presence or absence of an action. These semantically constrained recurrent units were called action proposal units since they only detected the presence or absence of an action but did not classify the proposed areas. The hidden states from these are passed on to another set of semantically constrained recurrent units where the classification task was performed. The length of the video and the number of actions performed were input to their network. Their work focused more on offline recognition which can not be useful for ego gesture recognition as discussed earlier. In [96] Zhang et al. approached this problem in a different setting. They argued that learning from trimmed action recognition can be transferred to untrimmed video action recognition. In this regard, they designed two networks with the same architecture, trained them separately on trimmed and untrimmed videos. Transferred the weights from trimmed video to untrimmed video using a knowledge transfer module (shown in Figure 3.14). Once the trimmed video network is trained, the parameters from intermediate fully connected layers are used in the transfer module. The Maximum Mean Discrepancy distance from analogous untrimmed network parameters is calculated and optimised over a loss in addition to the classification loss. However, their solution assumes that there is only one action per video, having such an assumption would not work in the case of egocentric gesture recognition.

Ego Gesture recognition from an untrimmed video is a sequence to sequence mapping represented by Equation 3.4.

$$l_t = F(I_t), t = 1 \dots T \tag{3.4}$$

where I_t is the input sequence of images, and l_t is the corresponding sequence of labels. Zhang et al. [9] solved the problem of ego gesture recognition using a combination of 3D CNNs to encode visual and motion aspects and then using a recurrent neural network to infer from the encoded features. This kind of approach is commonly used in action, and gesture recognition [95, 98, 99, 100, 101, 102, 103]. They used the 3D CNN + RSTTM proposed by Cao et al. [12] for encoding the visual and motion features and used LSTM for labelling the encoded features. However, since the number of frames without gestures are much higher than those with gestures they needed to use heuristics like including a certain number of frames and sub-sampling the videos with gestures to a certain length. These heuristics affect the network's performance, and figuring out the right combination is not a trivial task.

Kopuklu et al. [97] proposed using two separate networks, one to recognise if the frame contains a gesture or not and another to detect the gesture after accumulating the gesture frames recognised the first network. Their system maintains a queue which gets populated by frames detected as gesture frames. The crux of their algorithm relies on a postprocessing step which looks at the results from the detection queue and assigns a label to the entire queue in one step. They only solve partial labelling problem where they assign the gesture label after finding the beginning and partial completion of the gesture depending on a threshold.

3.3 Datasets

To train a generalisable deep neural network, we need large amounts of labelled data; we have already seen the reasons for this in Section 2.3. There have a wealth of Ego Centric Activity Datasets recently, The Something Something dataset [104], Epic Kitchens Dataset [105], Extended Georgia Tech Egocentric Activity Dataset(EGTEA) [106]. However, these datasets pose a different set of problems compared to recognising gestures from egocentric videos. They have interactions with objects which are not present in egocentric gestures. Another differentiating factor is recognition on these datasets are dependent on the background information too, so using these datasets to train and test egocentric gesture recognition networks is not viable. There are useful non-egocentric gesture datasets like IsoGD, ConGD [107]; however, like ego action datasets, they pose a different set of challenges compared to egocentric gesture recognition. We used NVGesture [73], AirGestAR [88], EgoGesture [9] datasets to train and test our network architectures in addition to the dataset we created. These datasets are reviewed below.

NVGesture dataset was created to enable gesture communication in a car. The setup for recording the video data included driving simulator environment where the person performing the gestures sat in the driving seat with a steering wheel(Figure 3.7). They defined 25 unique gestures that can be used for interfacing with control systems in an automobile. 20 different subjects performed the 25 defined gestures in 2 different sessions. The gestures were performed by using the right hand, while the subjects left hand was intended to control the steering wheel in the simulation environment. A SoftKinectic DS325 sensor was used to collect the colour and depth data, while a DUO 3D sensor was used to collect the stereo IR data. Though this dataset is not recorded from an egocentric perspective, it shares most of the features of an egocentric dataset. Unlike other non-egocentric datasets like IsoGD, ConGD [107], this does set captures mostly the user's hand movements, making it a useful dataset for testing egocentric gesture recogni-

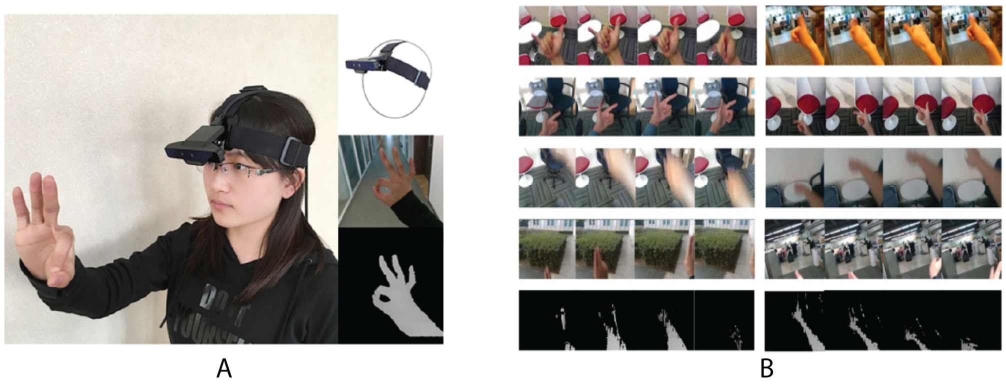


Figure 3.15: (A)EgoGesture [9] dataset capture setup, consists of an Intel RealSense SR300 mounted in a harness worn on a subject's head. (B) Some of the Gestures in the dataset, showing variations in style for the same gesture, variation in the background. The picture also illustrates some of the challenging scenarios with extreme motion blur, partial hand views, cluttered background.

tion networks.

Jain et al. addressed a lack of dedicated ego gesture dataset by publishing AirGestAR [88]. This dataset consisted of four gestures (Bloom, Click, Zoom-in, Zoom-out shown in Figure 3.11) performed by six different subjects of varying skin colours and lighting conditions. The data was recorded using a mobile phone mounted on a Google Cardboard box. Though the dataset consisted of ego hand gestures, the number of gestures were small, and the number of subjects performing the gestures was also small, there were no changes in the background, the environment was only indoors. Considering the above factors, the dataset was of limited use for training good networks for ego gesture recognition.

EgoGesture dataset [9] published by Zhang et al. was one of the first comprehensive ego gesture dataset introduced for training, testing and benchmarking different deep neural networks. In comparison to Jain et al.'s four gestures, they recorded 83 different static and dynamic gestures. Their dataset also included dynamic backgrounds, in both indoor and outdoor environments. 50 different subjects participated in collecting the dataset, giving it a wider variety of gesture durations and individual styles per gesture. Their

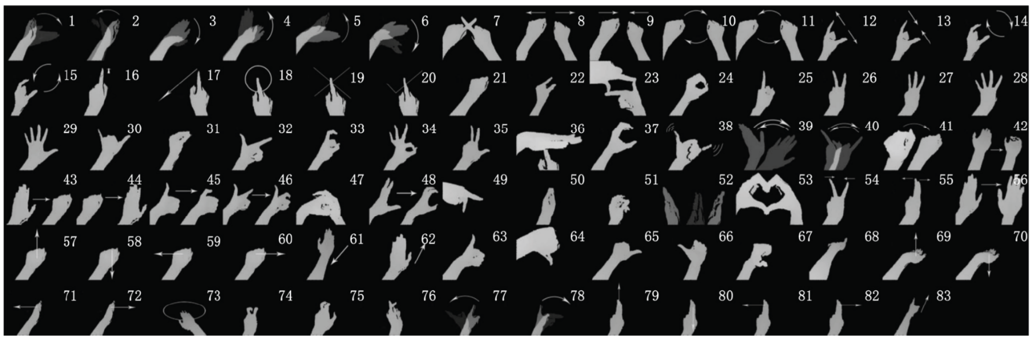


Figure 3.16: 83 different static and dynamic ego hand gestures in EgoGesture [9] dataset

setup consisted of Intel RealSense SR300 mounted on a harness that is worn on the user's head simulating an egocentric AR/VR device (illustrated in Figure 3.15

Data for two modalities depth and colour were collected at a framerate of 30 fps. Since both cameras are on the same device, the timestamps are synchronised aligning both the depth and colour videos. The gestures were performed in 4 indoor and two outdoor scenes, and this also included the subject being stationary or in motion while performing the gestures, adding more ego-motion to the videos to incorporate realistic scenarios. However, to add a new gesture to this dataset would require 50 subjects performing the gesture in a different environment to have a balanced dataset. We introduced the Green Screen Ego-hand Gesture dataset (Published work in ISMAR Adjunct 2018) to alleviate this problem.

3.4 Summary

In this chapter, we looked at various approaches to recognising both ego-hand gestures and non-ego-hand gestures from images. The approaches to solving ego-hand gesture recognition prior to deep learning were also discussed to get a broad perspective of research that was done. Furthermore, to gain an understanding and appreciation for using deep neural networks in comparison to the classical methods. The current available datasets, their advantages and limitations to train and test ego-hand gesture recognition networks were also reviewed.

The classical approaches suffered from limitations like using coloured gloves, keeping the hands stationary for few seconds in the beginning, stopping in between phases of gestures for identifying state transitions impeding the user severely to perform hand gestures naturally. Most of these limitations were overcome by the current deep neural network approaches. The state of the art approach to recognise ego-hand gestures from trimmed videos calculates data based convolutional features and compensate for the head motion through homography or affine transforms estimated using recurrent spatio-temporal transform modules. However, these homography or affine transforms do not pay particular attention to ego-hands in images which can potentially explain the difference in recognition accuracies between the stationary case (no ego-motion) and the non-stationary case (with ego-motion). To address this issue during recognition, the proposed approach discussed in Chapter 5 uses a network that can learn the spatial and temporal positioning of ego-hand while simultaneously recognising the gesture performed by the user.

In the case of recognition of ego-hand gestures from untrimmed videos, the current approaches copy the network architectures from trimmed videos, introduced heuristics to use the training data selectively. The use of such heuristics makes training the networks for recognising ego-hand gestures from untrimmed videos which were originally meant for trimmed videos extremely difficult. To avoid these difficulties and improve upon the current state of the art ego-hand gesture recognition on untrimmed videos an extension to LSTM with a novel loss function were proposed (explained concisely in Chapter 6).

The existing ego-hand gesture datasets used for training and testing recognition deep neural networks are hard to extend owing to the extensive number of times and different background environment a gesture needs to be performed by a user. The Green Screen

Ego-hand Gesture dataset discussed in the next chapter addresses this issue.

Chapter 4

Green Screen Ego-hand Gesture Dataset

We have established the need for a large dataset to train a generalisable deep neural network(Chapter 2.3). Thus, having a database with a large amount of different egocentric hand gestures like Ego-Gesture [9] is essential and also helps with the evaluation of different recognition algorithms. However, gestures are usually coupled with a specific task and are rarely the same across different applications. Defining a new gesture in the existing datasets like [9] is a cumbersome task since the new gesture needs to be performed by many subjects in many different scenarios. A useful dataset for training and evaluating deep neural networks must have a separate training, validation and test splits and correctly annotated labels. We present the additional characteristics required for a useful ego gesture dataset in the following section. Furthermore, the Green Screen Ego Gesture dataset and an augmentation method that alleviates the problem of data collection in different environments and manual annotation are presented in the later sections.

4.1 Characteristics of an Ego-hand Gesture Dataset

In addition to general characteristics needed for a good dataset to be useful for training deep neural networks, we defined some of the characteristics specific to handle ego-hand gesture recognition

- **Gesture style and duration**

  Each person performing gesture has a unique natural style and speed. A useful dataset needs to capture this variation giving the subject the flexibility to perform a gesture at their own pace and style.

- **Gestures using all combination of hands**

  Gestures can be performed with left or right or both hands. A useful dataset should have gestures performed by all the three combinations, so the networks trained can recognise all of them.

- **Skin colour**

  Human skin has a wide range of colours across the world population. Training a network with little variance of skin colour can create a heavy bias to a particular colour. A good ego-hand gesture training dataset needs to have every gesture performed by subjects of varying skin colours, to avoid the bias towards a particular skin colour.

- **Background**

  A network should be able to identify an ego-hand gesture oblivious to the environment it is being performed in. Whilst the background environment for recognising an ego-hand gesture itself is not essential, however, a dataset for ego-hand gestures must contain a wide variety of background environments including but not limited to combinations like indoors and outdoors, cluttered and uncluttered. Having such a variation in the background would help a deep neural network to differentiate an ego-hand from the background.

- **Lighting conditions**

  Images can be overexposed or underexposed, due to the movement of hands closer and farther away from the egocentric camera capturing the hand movements. So dataset must contain a good sample of images that overexposed, underexposed and sufficiently exposed.

- **Ego Motion**

  One of the challenges in ego-hand gesture recognition is the presence of ego-motion, which include head movement or movement of the subject or both. A good ego gesture dataset should contain gestures performed while stationary and during motion so that a network trained can rightfully isolate the hand movement with respect to changing background if there is any relative motion in the background.

With the characteristics required for an ego-hand gesture dataset defined, we look at the different gestures, data collection, annotation process of the Green Screen Ego Gesture dataset, and compare it to EgoGesture, AirGestAR datasets in terms of the defined characteristics they satisfy in the following sections.

4.2 Green Screen Dataset

One of the main limitations of existing large ego-hand gestures dataset [9] is the difficulty in extending it. A gesture needs to be performed by 50 different people in 6 different locations, to add a new gesture to this dataset. We proposed the Green Screen Dataset to alleviate this particular issue. A set of 10 gestures that use left, right and both hands (see Figure 4.1) were defined. The gestures defined were intended to be playful interactions in a video game. The data collection process was designed in such a way that it could

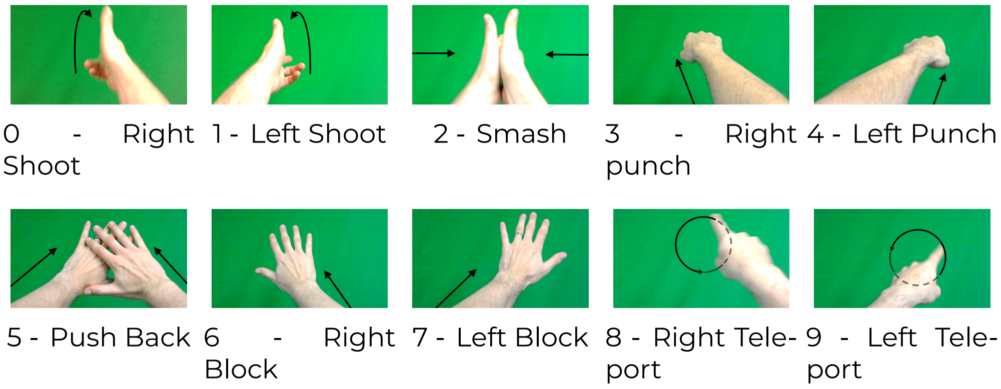


Figure 4.1: Our training set of 10 gestures captured in front of a green screen.

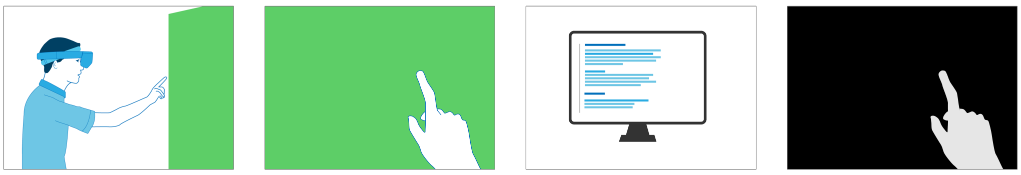


Figure 4.2: Data Collection Process for training set:: Step 1: User performs gestures in front of a green screen wearing HoloLens: Step 2: Record the images taken from Ego View camera in HoloLens which sees user's hand performing gestures and green screen background. Step 3: Transfer frame images to a computer and use a green screen extraction software to generate hand masks for each of the images. Step 4: Save the masks along with its corresponding RGB images.

capture most of the characteristics mentioned above if not all.

4.2.1 Data Collection

We collected **training gestures** from 20 users, each repeated the gesture three times. The users were performing these gesture in front of a green screen wore a head-mounted AR device, HoloLens in our case. The users were shown the ten gestures in advance to get acquainted with them. They performed these gestures without any restrictions on the duration or style of each gesture, allowing them to express naturally. This process resulted in a large variance in the duration per gesture and per user, which is described in Fig-

Database	# of Subjects	# of Backgrounds
AirGestAR	6	6
EgoGesture	50	8
GreenScreen	20	1

Table 4.1: Table comparing the number of backgrounds needed for existing ego-hand gesture databases with ours. Our database needs only one background compared to others making it much easier to collect data for adding new gestures.

ure 4.3. Since the users were also given the freedom to express their style, we could capture variations in style for each gesture, e.g. the ”shoot” gesture was performed with a single finger by few subjects, while other used two fingers. In addition to the images (RGB) captured by the egocentric camera, we also collected the 6DOF camera/head pose information that is given by the HoloLens. This collection of gestures performed in front of a green screen constituted our training dataset. The primary idea of such a training dataset is to replace the green screen with any required background decreasing the data collection burden. As noted in Table 4.1 we require only 1 background, namely green screen to collect training data whilst EgoGesture [9] and AirGestAR [88] require 8 and 6 different background correspondingly.

In addition to our training dataset we have generated **testing gestures** in natural settings, i.e. without green screen. These gestures were captured in real office environments with various backgrounds. We have collected testing gestures from 6 users, each gesture being repeated twice. After inspecting each video, we removed gestures that were performed outside the camera’s field of view (FOV) and we ended up with 7 to 9 samples per gesture.

4.2.2 Data Annotation

We provided two annotations for the training dataset. A hand mask annotation for every frame in which an ego-hand appears, and a label for each frame specifying the gesture if it contains any. The hand mask annotation was created automatically by processing the

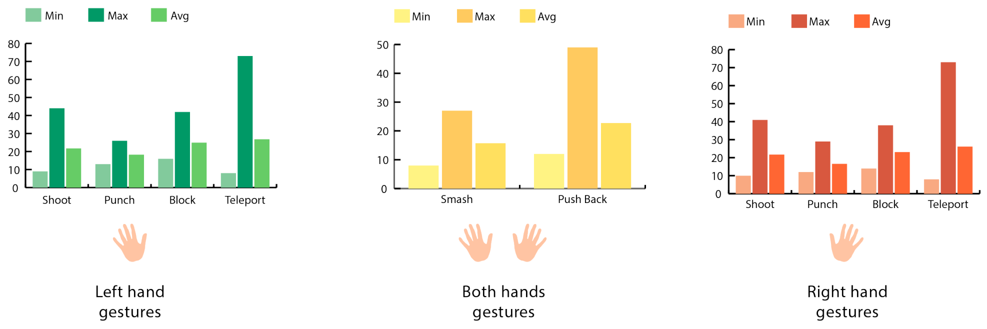


Figure 4.3: Bar graphs describing the variance in the number of frames the user needed to perform a gesture. The graph in green corresponds to left hand gestures, yellow both hands and red to right hand.

recorded images in the training set using a green screen segmentation algorithm of a typical video editing software (Natron), eliminating the need for manual generation of hand masks per image. Figure 4.2 illustrates the process of database generation. The hand masks, along with their corresponding images and labels per frame were stored. Besides, we also collected and stored the ego position generated by HoloLens per frame. Our dataset is generated from users with varying skin tones, under different lighting conditions, some users wearing full sleeves, and some wearing watches or bracelets, to reflect real-world situations. In comparison, in previously captured datasets (i.e. [9, 88]), the gestures are more clinical in a sense that each gesture has the same movement of hands or restricted duration. Unlike others, we showed users a video of each gesture at the beginning of the capture and then let them express the gesture naturally.

4.3 Summary

This chapter introduced the Green Screen Ego-hand gesture dataset, which was created to alleviate the problem of large scale data collection for training ego-hand gesture recognition deep neural networks. The data collection and annotation strategy, the differences

	Style and Duration	**Hand Combinations**	**Skin Colour**	**Background**	**Ligthing Conditions**	**EgoMotion**
GreenScreen	✓	✓		✓	✓	✓
EgoGesture [9]	✓	✓		✓	✓	✓
AirGestAR [88]	✓					

Table 4.2: Table listing the three ego-hand gesture databases and the characters defined in Section 4.1 they adhere to are represented with a ✓

between training and testing sets were explained in detail. The properties that would make a good ego-hand gesture dataset were discussed. The Green Screen dataset was compared to two other publicly available ego-hand gesture datasets in terms of the properties these datasets adhere to and the difficulty involved in adding new gestures to them. In the following chapters novel deep neural network architectures, a new loss function and evaluation metric for ego-hand gesture recognition, trained and tested on various available ego-hand gesture datasets including the Green Screen dataset is introduced.

Chapter 5

Ego-hand Gesture Recognition on Trimmed Video

Ego-hand gestures can be used as interfaces in head-mounted AR/VR devices to interact with the virtual elements, as they are a natural way for humans to communicate. We have seen various ego-hand gesture recognition algorithms that utilised classical methods of calculating handcrafted features with various techniques like SVMs, HMMs and also more current methods which are based on deep neural networks, and their limitations in Section 3.2. In this chapter, the work on recognising ego-hand gestures on trimmed video using simultaneous segmentation is explained. The idea of 'Simultaneous Segmentation and Recognition' is presented first followed by the deep neural network architecture and experiments and results performed on the Green Screen Dataset(Chapter 4) and Ego-Gesture [9], AirgestAR [88].

5.1 Simultaneous Segmentation and Recognition

The idea of segmenting ego-hands from images recorded through head-mounted devices and calculating features on the segmented images to recognise egocentric activity has been employed in various works like [108, 109, 110]. In all these approaches, variants of convolutional neural networks are used to create a segmentation of ego-hands, and these segmentations are separated. The separated segments, along with some handcrafted features, are used in a disjoint neural network to recognise the activity being performed in the input video. In Simultaneous Segmentation and Recognition, this problem is approached by attempting to find embeddings that simultaneous help recognises a gesture in a sequence of images and also the ego-hand segmentation map. This idea is formalised below.

Let I_s be the sequence of images containing ego-hands that we want to recognise an ego gesture from and l its corresponding class label. The problem of ego gesture recognition can then be defined as finding a function f, such that it maps the given image sequence I_s to l as described in equation 5.1.

$$f(I_s) = l \tag{5.1}$$

So far the approach to finding the function f using hand segmentation has been (represented by equation 5.2), to find three functions g, h, k where

- g takes in a sequence of images I_s, produces a sequence of ego-hand masks M_s.
- h takes in the sequence of ego-hand masks M_s generated by g, extracts a set of features X_s, where $dim(X_s) \ll dim(I_s)$
- k takes in the set of features X_s generated by h, and maps it to the corresponding class label l

$$f(I_s) = g(I_s) \circ h(M_s) \circ k(X_s) \tag{5.2}$$

We rephrase this problem as finding the feature set E_s for the sequence of images I_s which can be simultaneously used for finding their corresponding segmented hand masks M_s and also the class label l. In our approach, we define 3 functions a, b, c, where

- a takes in a sequence of images I_s, produces a feature set E_s, where $dim(E_s) \ll dim(I_s)$
- b and c take in the feature set E_s, and simultaneously produce ego-hand masks M_s and class label l.

Equation 5.3 summarises the concept of simultaneous segmentation and recognition. One advantage we have over previous methods is that, once the functions a, b, c are estimated, we do not need the function b that generates ego-hand masks for recognising the class label.

$$f(I_s) = a(I_s) \circ c(E_s) \ni b(E_s) = M_s \tag{5.3}$$

We used deep neural networks to design our functions a, b, c. Autoencoders [44] are well studied and known for finding a reduced representation of the input. Thus we used them for designing functions a, b with a modification. In general, the input and the output for an autoencoder are the same. However, we use an RGB image with ego-hand as input and a segmented ego-hand mask as an output. This strategy helps our network to find a reduced representation of ego-hand masks from the input RGB images.

There are two components to an autoencoder, an encoder network which encodes the input to a short, intermediate feature vector and a decoder network which recovers the output from this intermediate representation. In our case function a represents the encoder part of the network and function b represents the decoder part of the network. We use the embedding generator and the LSTM as the function c, which also takes in the intermediate feature vector

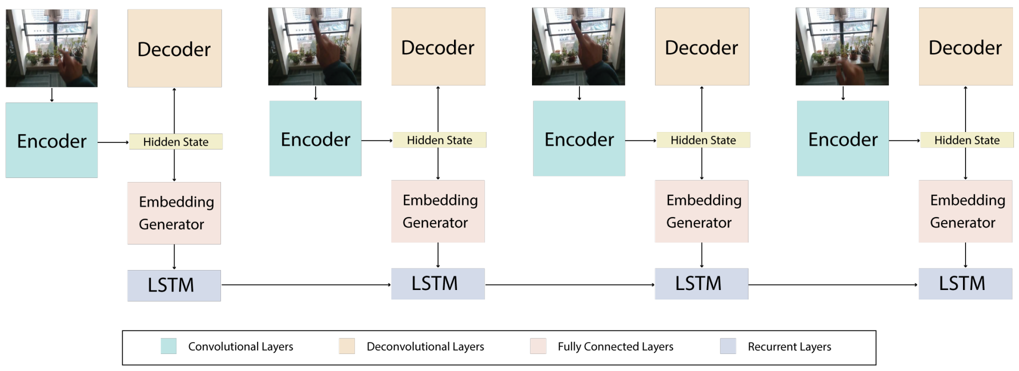


Figure 5.1: Our network architecture. The Decoder outputs a segmented ego hand map per image in the sequence. This part of the architecture is not needed once the network is trained. The sequence of images converted to embeddings corresponding to ego hand segmentation is used by LSTMs to recognise the ego gesture.

as input and generates the class labels for sequences of images. We elaborate on the network architecture of each component in Section 5.2. The end-to-end training for simultaneous segmentation and recognition is explained in Section 5.3.

Our network design and training approach yielded a considerable improvement compared to state of the art on EgoGesture [9], AirgestAR [88] and also Green Screen Dataset. The details of the network architecture, training and experimental results are discussed in the following sections.

5.2 Network Architecture

Our network architecture consists of two main components (Figure 5.1), an autoencoder like network that generates segmented ego-hand images and embeddings for LSTMs, and LSTMs that feed on this component to recognise the corresponding ego gesture. An autoencoder is a neural network that is intended to reproduce the input. Internally it reduces the input to a hidden state whose dimensions are lower than that of the input. The lower-dimensional hidden state is used to replicate the input. However, in our architecture, instead of replicating the input, we generate a segmented ego-hand image with the same dimensions as the input. This output maps each pixel in the input image to either ego-hand or not. Besides, we also output the embedding generated from the hidden state, whose dimensions are much smaller than the input image. Though there exists other networks like the U-Net [111] and its variants for segmentation, one of their main features are the shortcuts that occur from higher-level convolution to high-level deconvolution layers. In such a scenario, there is a loss of information that would not be passed down to the hidden layer that generates the embedding that we use to infer the ego-hand gestures. Networks like these can use be more useful for ego-hand segmentation, but not for gesture recognition, so autoencoder network is preferred.

For the encoder, we used the first two layers in ResNet18 [112]. Further, we add more convolutional layers, progressively decreasing the size of feature maps by setting the stride to 2 and simultaneously increasing the number of features to n, also the size of generated embedding. This size n is a hyperparameter that is set depending on the complexity of the recognition problem. For a dataset like EgoGesture with a large number of gestures n is larger and for dataset sets like Green Screen and AirGestAR, n can be set to a smaller size. Each of the convolutional layers is followed by batch normalisation and ReLU layers.

The decoder has a series of deconvolutional filters, takes in the hidden state generated by the encoder as input, upsamples the features back to the size of the input image and simultaneously decreases the number of features. The final deconvolution layer outputs a two-channel image with the input size. The two channels in the output contain the probability of the pixel belonging to ego-hand and the background.

Depending on the complexity of the data, additional fully connected layers are added to the generated hidden state, before feeding as an input to LSTM layers. LSTMs have been widely used for video classification [113], action recognition [114, 115] and gesture recognition [73, 12]. In all the approaches above, the embeddings provided to LSTMs as input play a vital role in determining the recognition accuracy. In our architecture, embeddings based on segmented ego-hands as described in Section 5.1 provide inputs specific for ego-hands to LSTMs, leading to better recognition accuracy.

5.3 Training Strategy

Three datasets Green Screen Dataset, AirGestAR, EgoGesture, were used for training and testing the network. These three datasets though intended for ego-hand gesture recognition, are different from each other. EgoGesture's intended purpose is to train and benchmark recognition deep neural networks, while Green Screen dataset was introduced as a way to decrease data collection. Furthermore, AirGestAR was a small dataset which can be used for a first pass evaluation since it had only four gestures and the benchmark accuracy scores when it was introduced were already at around 93%. Since these three datasets are very different though the overarching network architecture remained the same, it had to be tuned for each of them differently. However, the following steps (as shown in Figure 5.2) were followed for the datasets where ego-hand masks were available.

1. Train the AutoEncoder when ground truth ego-hand mask are available (Fig5.3).
2. Train the AutoEncoder + Classifier on individual images by connecting the hidden state to SoftMax Classification (Fig5.4).
3. Train the LSTMs using the intermediate hidden state representation as input (Fig5.5).
4. Train the entire network together using the weights from the previous steps to initialise the weights (Fig5.6).

Cross Entropy loss which is used for training all the phases, is calculated using the equation 5.4. Where n is the number of classes,

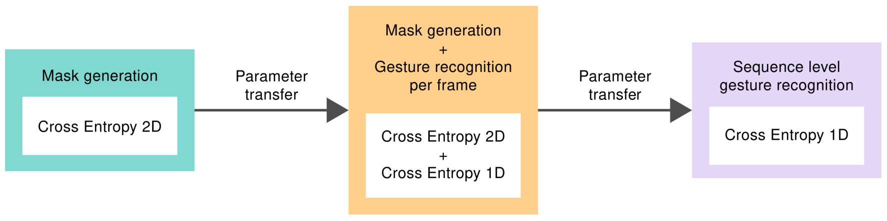

Mask generation
Cross Entropy 2D
Parameter transfer
Mask generation + Gesture recognition per frame
Cross Entropy 2D + Cross Entropy 1D
Parameter transfer
Sequence level gesture recognition
Cross Entropy 1D

Figure 5.2: Parameter transfer from various phases of training.

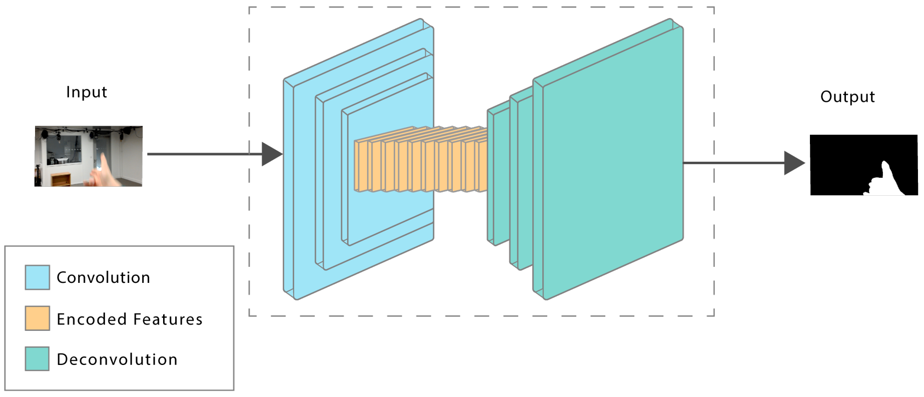


Figure 5.3: **Phase 1**: In the first phase we train the ego-hand Auto Encoder network. This part of the network consists of the Encoder which encodes RGB images into hidden states and the Decoder that decodes the ego-hand masks from these hidden states. At this stage of training the order of images does not matter, since we create ego-hand mask from its corresponding single RGB image. CrossEntropy Loss was used to train the AutoEncoder network.

$y_i = 1$ if i is the ground-truth class label, else $y_i = 0$ and p_i is the probability output for class i by the network after SoftMax layer.

$$L_{ce} = -\sum_{i=1}^{n} y_i log(p_i) \tag{5.4}$$

The training strategy and parameters for each dataset are discussed below.

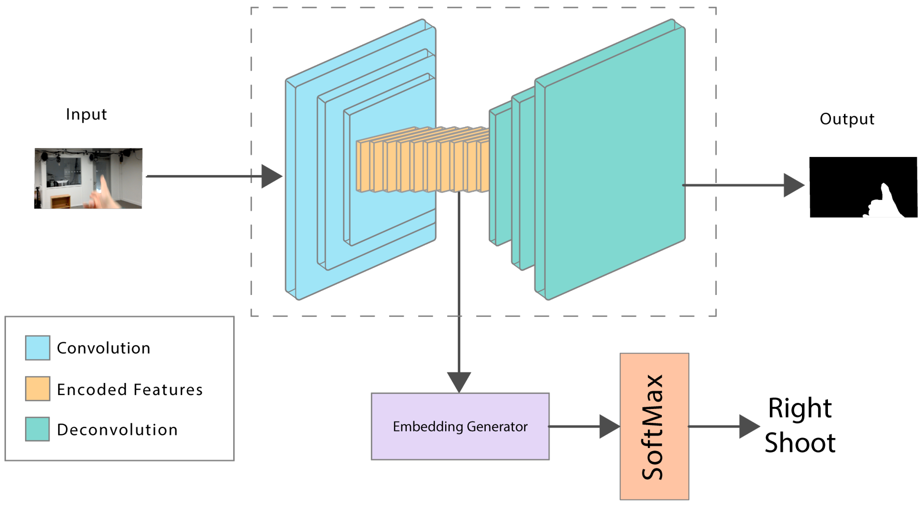


Figure 5.4: **Phase 2**: In the second phase, the hidden state from the AutoEncoder in Phase 1 is connected to an embedding generator. This embedding generator is then connected to a Softmax classifier to recognise a gesture in individual RGB images. The sum of CrossEntropy loss for Ego-hand mask and CrossEntropy Loss for gesture label is used to train the network in Phase2.

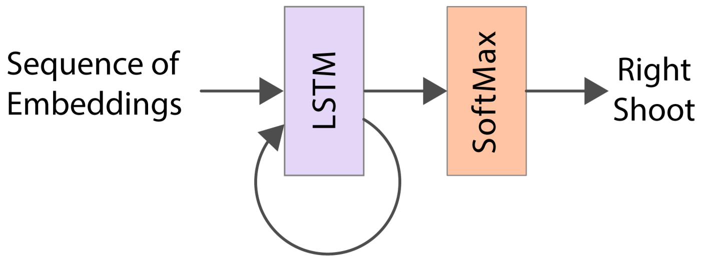


Figure 5.5: **Phase 3**: A sequence of embeddings are created for corresponding sequence of RGB images using the network from Phase 2. These embeddings sequences with corresponding gesture labels are used to train a LSTM with Softmax Classification. Training the LSTMs in this manner gives us the ability to train with large batches resulting in better generalisation capabilities. CrossEntropy loss for gesture label is used to train the LSTM.

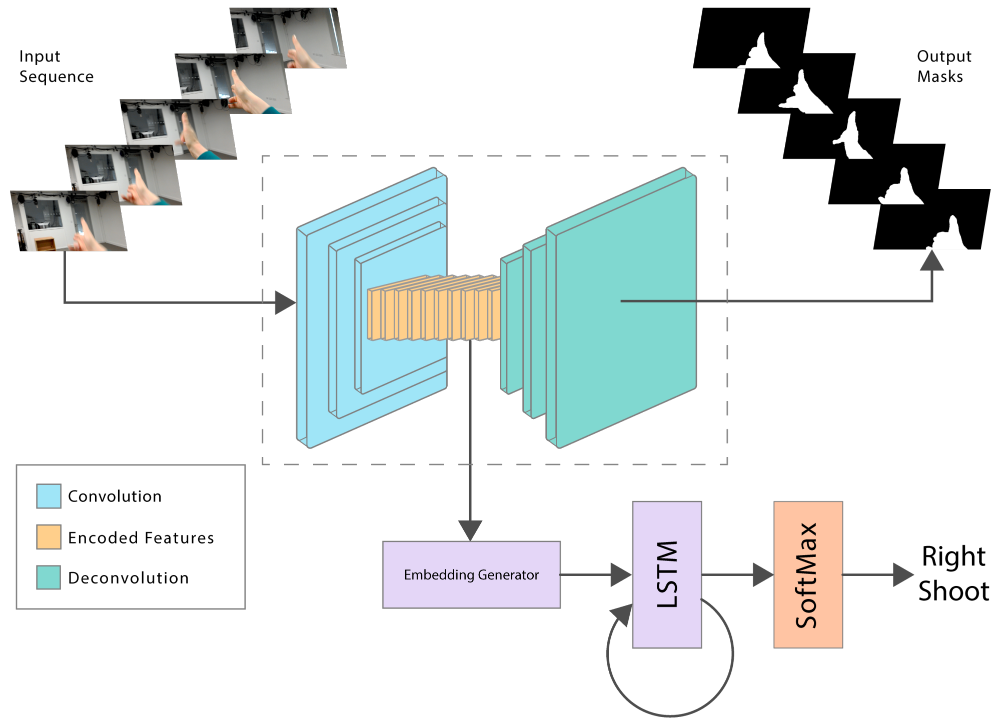


Figure 5.6: **Phase 4**: In the final phase, all the weights for different parts of the networks trained in previous phases are used to initialise the full network. And the entire network is trained with sequence of images and their corresponding labels to fine tune the network. Since the individual network parts are already trained, we could train the network with a sum of three CrossEntropy losses used in the previous phases, without any need for weighting.

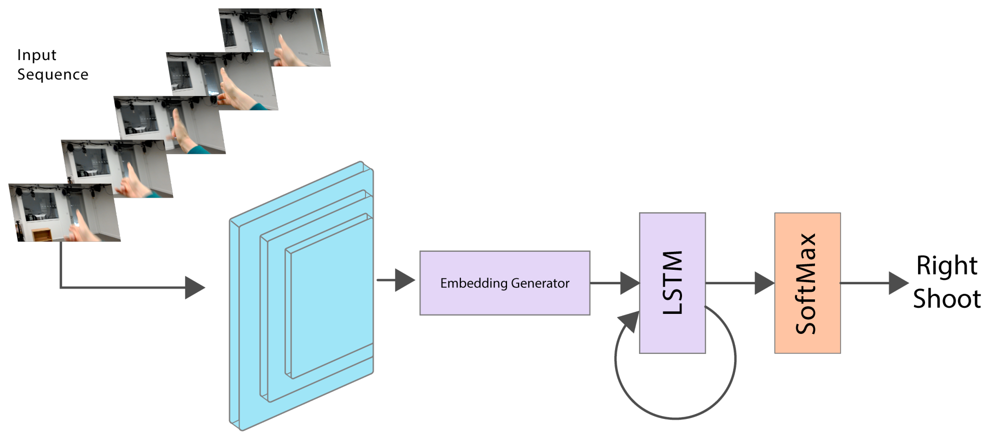


Figure 5.7: **Inference**: Since the primary aim of our network is to recognise a gesture in Trimmed video, we do not need the decoder during inference. As illustrated the encoder is directly connected to the embedding generator.

5.3.1 Green Screen Dataset

The Green Screen dataset as the name suggests contains egocentric views of hand gestures performed in front of a green screen for the training set. The primary purpose of this dataset as discussed in Chapter 4 is to allow easier addition of new gestures. The training dataset needs to be augmented with real-world images to learn to discern ego-hand from the background. This process of augmenting the training dataset is discussed next followed by the training strategy used for this dataset.

Data Preprocessing

The data preprocessing step for preparing the training dataset involves replacing the green screen with a real-world image. To perform this using mask that was obtained by the green screen removal process as described in Section 4.2.2 and the corresponding egocentric image and a natural background image are combined to replace the green screen with the natural background image. The following Algorithm 1 summarises the process of data augmentation.

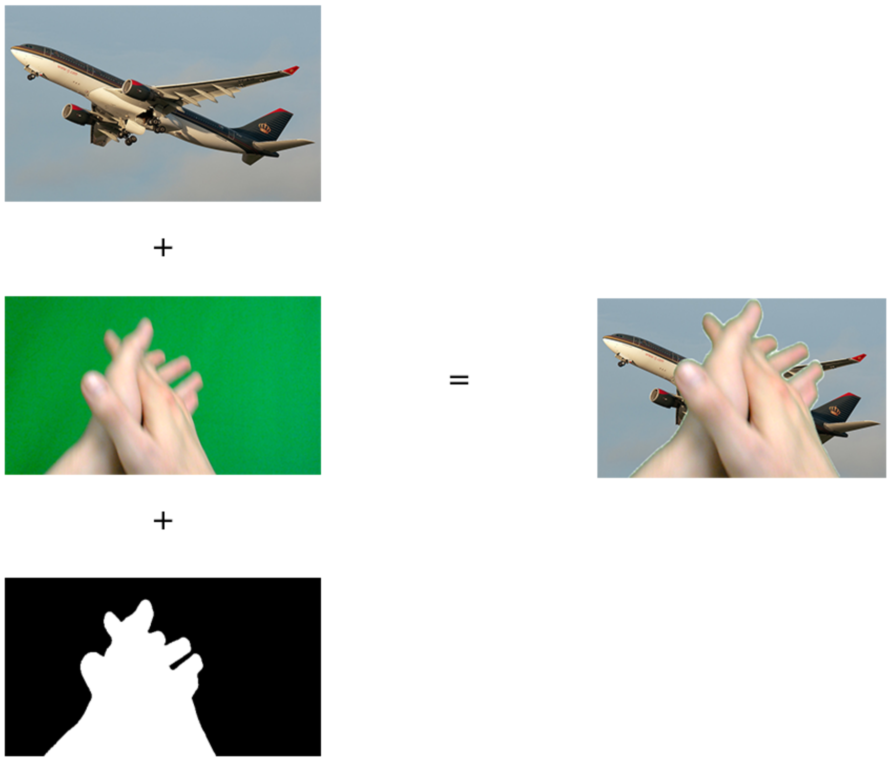


Figure 5.8: Data augmentation step. The training image on the right is a combination of a random background and a segmented frame using a binary mask (on the left).

The Figure 5.8 shows the mask applied to one of the images. This process creates n images from one captured image with the same mask. As background, we chose n random images from a set of 40,000 images from the COCO Test dataset [116]. For our training, we set $n = 5$, which increases the size of the dataset fivefold. In addition, we also add one of the following *'none'*, *'poisson'*, *'gaussian'*, or *'salt&pepper'* noises randomly, to ensure that we are not over-fitting data. Then, we store each of these images separately along with the gesture id and their corresponding mask. We scale down all the images and masks to 224×126 resolution and normalise them.

Algorithm 1 Data Augmentation for the training set in Green Screen Dataset.

```
1: $n = 5$    ▷ Initialise to the value of desired increase in dataset size, 5 is chosen for the experiments.
2: $B = B_1, B_2, \ldots B_n$ be the set of background images
3: $G = G_1, G_2, \ldots G_m$ be the set of green screen training image
4: $M = M_1, M_2, \ldots M_m$ be the set of corresponding masks.
5: $T$ = set of augmented training images    ▷ initialised to empty set.
6: for $g_i, m_i \in G, M$ do
7:     $b$ = set of $n$ random images from $B$
8:     $j = 1$
9:     $t$ = set of training images for $g_i$
10:    $x$ = Randomly chose a noise type from (none, poisson, gaussian, salt & pepper)
11:    for $j \leq n$ do
12:        $t_j = b_j$
13:        replace all pixels in $t_j$ with pixels from $g_i$ where $m_i \neq 0$
14:        Add $x$ to $t_j$
15:        append $t_j$ to $t$
16:        $j = j + 1$
17:    append $t$ to $T$
18: return $T$
```

Training Strategy

The training data is 90/10 split for training and validation, respectively. The network is trained in 3 phases; parameters from each phase are transferred to the next one(Figure 5.2). In the first phase, the autoencoder is trained with one loss function appended to deconvolution layers learning the ego-hand masks. 2D cross entropy loss and ADAM optimiser with learning rate 10^{-5} are used to train during Phase 1. The data is shuffled and trained for five epochs with a batch size of 50. All the parameters used are summarised in Table 5.1.

	Input	Layers	Output
Encoder	RGB Image (224x126)	inp, out, size, stride, padding Resnet18 Layer1 Resnet18 Layer2 Resnet18 Layer3 Conv2D(128, 128, 3, 2, 1), BatchNorm, ReLU Conv2D(128, 64, 3, 2, 1), BatchNorm, ReLU	Hidden State

Decoder	Hidden State	inp, out, size, stride, padding
		Deconv2D(64, 32, 4, 2, 1)
		Deconv2D(32, 16, 4, 2, 1)
		Deconv2D(16, 8, 4, 2, 1)
		Deconv2D(8, 4, 4, 2, 1)
		Deconv2D(4, 2, 4, 2, (2, 1))
			EgoHandMask

Embedding Generator	Hidden State	inp, out
		AvgragePooling(7x4x64, 1x1x64)
		FullyConnected(64,10/4)
			Embedding

LSTM	Embedding	inp, hidden, layers
		LSTM(64, 128, 3)
		FullyConnected(128, 10/4)
			Class Label

Table 5.1: The hyperparameters used in various components of SSAR network trained for Green Screen and AirGestAR Datasets.

Phase	**Optimiser**	**Loss Function**	**Batch Size**	**Epochs**
Mask Generation	Adam, 10^{-5}	2D Cross Entropy	50	5
Mask Generation **+** **Frame Level Recognition**	Adam, lr=10^{-6}	2D Cross Entropy + 1D Cross Entropy	50	18
Sequence Level Recognition	SGD, lr=10^{-6}, momentum=0.7	1D Cross Entropy	1	60

Table 5.2: Parameters used in various phases of training. Slow learning rate in Phase 1 provided most stable training error reduction. Phase 2 learning rate is smaller than phase 1 since the encoder and decoder are already trained.

In the second phase, an average pooling layer is appended to the embedding generated by the encoder and then a fully connected layer with outputs size N_g(number of gestures in the dataset, 10 in the case of Green Screen Dataset) . A 1D Cross Entropy loss is added to do per frame gesture recognition. The parameters obtained from Phase 1 are transferred to Phase 2. Then the network is trained on a combined loss function using 2D cross entropy loss from phase 1 and 1D cross entropy loss from this phase with equal weight for both loss functions. We use ADAM optimiser with learning rate 10^{-6} and train for 18 epochs with batch size of 50. At this point a frame level gesture recognition is obtained, which is inaccurate. The inaccuracies stem from the fact that a single frame only captures the pose of hand at a moment in time which can be the same across many gestures. However, a sequence of images can rightly identify a gesture, and phase 3 is used for this.

For the final phase, the data augmentation approach is modified slightly. Instead of using a random background and noise for every image, we now use the same random background and noise for the whole gesture sequence, such that each of the sequences has the same background and noise. The embeddings generated by the autoencoder after Phase 2 training are stored on disk per gesture sequence. The saved sequences are as input to the LSTM. The hidden layer from the last sequence is connected to a fully connected layer with output size N_g and then to a 1D cross entropy loss for sequence recognition. All the parameters from earlier phases are used to initialise the weights of the entire network together, and a final end-to-end training combining the three loss functions is performed. For optimisation, we use a Stochastic Gradient Descent algorithm, with a learning rate of 10^{-6} and momentum 0.7. We train for 60 epochs. All the hyper-parameters used for training are summarised in Table 5.2. After this final phase, we get our full network for sequence gesture recognition.

5.3.2 AirGestAR

Unlike the Green Screen Dataset or the EgoGesture Dataset, AirGestAR does not have hand mask annotations for the training data. To avoid manual mask creation, the Phase 1 network that was trained with the Green Screen Dataset was to used generate these masks automatically. After initial visual verification, we used these masks as ground truth in addition to the frame-level labels in the Phase 2 training. Finally, we followed the same procedure mentioned in Section 5.3.1 for Phase 3 training. The number of parameters that need to be estimated can approximate the complexity and size of a network. All the parameters used are summarised in Table 5.1. The proposed network compared to the network by Jain et al. [88] is much smaller but yields better results under the same experimental setup(Table 5.3 lists the total number of estimated parameters for our network in comparison to the AirGestAR network).

	# of Parameters
SSAR	960485
AirGestAR	17535551

Table 5.3: Size of **SSAR** network in comparison to AirGestAR [88] in terms of number of parameters to be estimated.

5.3.3 EgoGesture Dataset

In the EgoGesture Dataset, there are no explicit hand masks provided. However, the depth images provided were used to create binary ego-hand masks by thresholding them. The ego-hand masks created this way were used as ground truth annotated data for Phase 1 of training the network. The first phase consisted of training the encoder, decoder and embedding generator together to output the segmented ego-hand image and the input to LSTMs. Unlike the scenario for Green Screen and AirGestAR datasets, simple average pooling did not provide enough expressive power for the embeddings generated by the encoder. Two fully connected layers decreasing the final embedding size to 84 were introduced. In this training step, each image is considered an individual entity. We shuffle all the images with ego-hands without respecting their order in a video, split the total images into training, validation and testing sets. We use two loss functions, one to control the generation of segmented ego-hand images and another to label each embedding with the corresponding gesture. Cross-entropy loss for segmented ego-hand images and label loss with equal weights are used for backpropagation.

The LSTM layers are trained in the second step to recognise ego-hand gestures from sequences of embeddings. The videos each containing a single gesture are split into training, validation and testing sets. Embeddings for videos are generated using the network weights from Phase 1. These embeddings are then used to train the LSTMs. Isolating LSTM training allowed the usage of bigger batch sizes which helped in better generalisation and also to use sequences of arbitrary length. Cross entropy loss is used on the final fully connected layer to classify each sequence.

In the final step, we train the entire network together, end-to-end by connecting the embedding generator branch to LSTMs as input. We initialise the encoder, decoder, embedding generator with weights from step 1 and LSTMs with weights from step 2. The final loss function is the sum of Cross-Entropy label loss from LSTMs and segmented ego-hand images loss from the decoder are. All the hyperparameters are summarised in Table 5.4.

The experiments and results in comparison to state of the art on three datasets are reported in the following section.

	Input	Layers	Output
Encoder	RGB Image (224x126)	inp, out, size, stride, padding Conv2D(3, 64, 7, 2, 3), BatchNorm, ReLU Resnet18 Layer1 Resnet18 Layer2 Conv2D(128, 128, 3, 2, 1), BatchNorm, ReLU Conv2D(128, 256, 3, 2, 1), BatchNorm, ReLU	Hidden State

Decoder	Hidden State	inp, out, size, stride, padding
		Deconv2D(256, 64, 4, 2, 1)
		Deconv2D(64, 32, 4, 2, 1)
		Deconv2D(32, 16, 4, 2, 1)
		Deconv2D(16, 8, 4, 2, 1)
		Deconv2D(8, 2, 4, 2, (2, 1))
			EgoHandMask

Embedding Generator	Hidden State	inp, out FullyConnected(7168, 2048), BatchNorm, ReLU FullyConnected(2048, 83)	Embedding

LSTM	Embedding	inp, hidden, layers
		LSTM(83, 83, 4)
		FullyConnected(83, 83)
			Class Label

Table 5.4: The hyperparameters used in various components of SSAR network trained for EgoGesture Dataset. Since EgoGesture dataset has more gestures than Green Screen and AirGestAR the hidden state size is set to 256 instead of 64. This allows the encoder to represent more poses that are needed to discriminate between the large set of gestures.

5.4 Experiments and Results

The deep neural network architecture proposed after following the training strategies discussed in Section 5.3 is tested on all the three datasets. Training and testing for the network were implemented in PyTorch. PC with an Intel Core i7 CPU and NVidia Titan Xp GPU was used for hardware. Test results, ablation and validation studies on each of these datasets are reported in the following sections.

5.4.1 Experiments on Green Screen Dataset

	# of Gestures Classified Correctly	**# of Gestures Classified Wrongly**	**Accuracy %**
Frame Level	49	35	58.33
Sequence Level	60	24	71.42

Table 5.5: Accuracy results for gesture recognition on the Green Screen Dataset.

The test set for Green Screen Dataset contains ten natural gestures. There are several challenging scenarios for recognition where fingers are clipped, frames have a strong motion blur, and there is a variation of style within a gesture (see Figure 5.9 for examples). As mentioned in Section 5.3.1, the network was trained in three phases. The results from phase two and three of training are reported since phase one training involved finding ego-hand masks, and that is not the primary objective, no comparative studies were done for this phase.

The network result obtained from phase two of training can perform recognition per frame. The input to this network is a sequence of RGB images containing a gesture, and we get a gesture label prediction for each frame. A simple voting strategy was followed, giving each gesture a vote if a frame is predicted to be that gesture. The sequence is then assigned the label with the maximum number of votes. The process is labelled as *frame level* recognition.

For *sequence level* recognition, we input a sequence of images to the network from Phase 3. As we can observe from the results in Table 5.5 sequence level recognition performs much better than frame level. The same hand pose can be part of multiple different gestures but during different stages of performing the gesture. Since frame-level recognition does not consider time, it could easily classify a gesture incorrectly. Adding a temporal recognition component like an LSTM solves this issue as is evident from the results in Table 5.5.

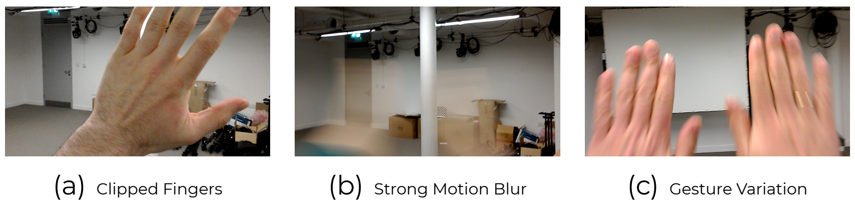

(a) Clipped Fingers (b) Strong Motion Blur (c) Gesture Variation

Figure 5.9: Examples of frames from testing sequences. **a)** Left Block gesture that was not performed fully inside the camera FOV. **b)** Smash gesture done at high speed with a strong motion blur effect. **c)** Push Back gesture that is a variation to the one in the training dataset.

To analyse recognition performance on each gesture we present a normalised confusion matrix in Figure 5.10 for results from the *sequence level* recognition. The mislabelled gestures are within the same hand (as in a left-handed gesture is being labelled as another left-handed gesture but not a right-handed). The recognition of gesture 7 - Left Block is especially low and is confused with Left Shoot and Teleport gestures. Looking at the testing videos closely, one observation that could explain this confusion is a large head movement that creates relative motion inside the frame similar to the one in Shoot (up-left motion) and Teleport (circular motion in left direction). To improve the accuracy in these situations, we are planing in the future to utilise the head pose transformation that can be obtained from HoloLens.

Also, in the Teleport gesture, some users used the whole arm to create circular motion, where others used only one finger. This small finger-motion is especially challenging to distinguish from an egocentric view as it can be occluded by the arm or hand, and can be easily confused with Shoot or Punch gestures.

The network to find hand pose [87] which was used in AirGestAR [88] could not recover poses for many of the ego-hand images in our dataset. It was not designed to handle complex scenarios like motion blur and clipped fingers which frequently occur in our dataset. Our network, however, could handle such situations which was not the case with AirGestAR network strategy. So we could not perform a comparative study using their network.

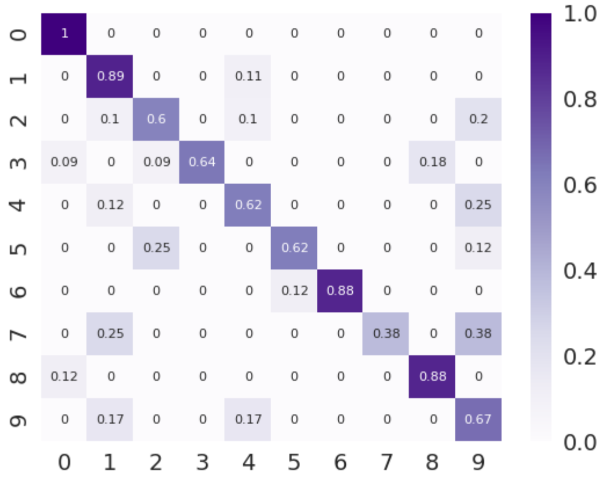


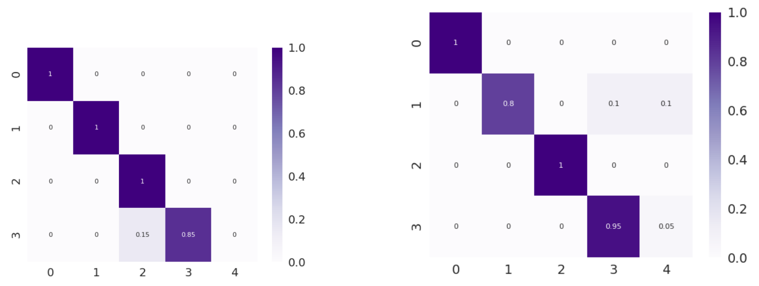
Figure 5.10: Normalised Confusion Matrix for 10 gestures in our database. X-axis has the predicted labels and Y-axis the ground truth labels

5.4.2 Experiments on AirGest Dataset

The AirGestAR dataset, as discussed in Section 5.3.2, does not have ground truth ego-hand mask annotations, which are necessary to perform the first phase of training. In order to avoid creating the ego-hand masks manually, the network trained with Green Screen Dataset was used to generate ego-hand masks automatically. After visual verification, we used these masks as ground truth in addition to frame-level labels in phase two training. The final phase of training and testing are reported. To provide comparative results, we used the same training and testing data, as described in [88]. The confusion matrix is presented in Figure 5.11 and the overall accuracy in Table 5.6. Our final network's performance can match the AirGestAR network's despite being orders of magnitude smaller in size(see Table 5.3 for reference).

	Gestures Classified Correctly	**Gestures Classified Wrongly**	**Unclassified**	**Probability Threshold** (σ)	**Accuracy %**
SSAR	77	3	0	0.5	96.25
SSAR	75	3	2	0.66	93.75
AirGestAR	75	2	3	0.7	93.75

Table 5.6: Accuracy results for SSAR network and network from [88] on AirGestAR dataset. Following the procedure used in [88] we experimented with different probability thresholds to mark a gesture sequence as classified vs unclassified. Any sequence with probability lower than the mentioned threshold is marked as unclassified.



Ours with $\sigma = 0.5$ AirGest $\sigma = 0.7$

Figure 5.11: Confusion matrix for AirGest dataset.

5.4.3 Experiments on EgoGesture Dataset

EgoGesture dataset is the largest and most comprehensive Ego-hand gesture dataset publicly available for benchmarking and testing recognition deep neural networks. Hence, the SSAR network was tested and validated thoroughly on this dataset. The dataset contains untrimmed videos of subjects performing the gestures. The segments of videos containing only ego-hand gestures are used for training, validation and testing.

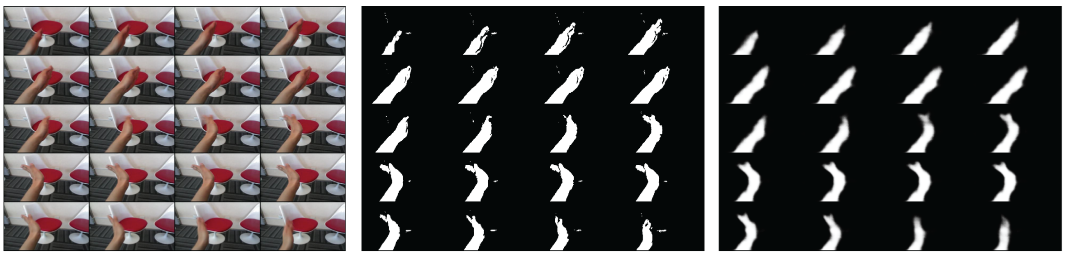

RGB Images
Ego hand segmentation created from depth data for ground truth
Ego hand segmentation generated by our network

Figure 5.12: EgoGesture Dataset [9] does not provide explicitly annotated hand mask data. We threshold depth maps provided to extract pixels responsible for ego hands to generate ground truth data. This process can introduce noise, but our network learns to generate segmented ego hands without noise.

The encoder, decoder and embedding generator are trained considering the images to be independent and not part of a sequence. The images with ego-hand are divided into training, validation, testing sets with 0.6, 0.2, 0.2 splits yielding $536938, 178979, 178978$ images respectively. The ResNet18 layer 1, layer two are initialised with pre-trained ImageNet weights; the rest of the weights and biases are initialised with zeros. We used a batch size of 100, set the learning rate to $1e-6$ and used ADAM optimiser. The training is done until the validation accuracy does not improve. The test set accuracy was used as an intermediary validation step. Figure 5.12 shows some ego-hand segmentations generated by our decoder after training the encoder and decoder. On close inspection of Figure 5.12, it could be noted that though the masks predicted are not 100% aligned with ground truth, and they are sufficient to improve the accuracy of ego-hand gesture recognition. Once the encoder, decoder and embedding generator are trained and tested, the embeddings are generated and stored on the disk. Unlike, encoder, decoder and embedding generator, training the LSTMs requires sequential data. So the embedding is generated per sequence of images that belong to one gesture.

Of the various initialisation techniques available for LSTMs in the second stage of training and a combination of orthogonal [117], and Xavier normal [118] initialisation for input and recurrent weights, and zeros for biases worked best for convergence. The embeddings generated and stored per input video are used in this phase to train the LSTMs. Unlike the state of the art [12], whose network is limited to identifying gestures from a sequence of length 40, this step gave the ability to train with arbitrary sequence length and large batch size. The batch size was set to 100 with padded sequences for training the LSTM layers. The input size and hidden size for LSTM layers was set to 83. The segmented ego-hand gesture videos were divided into training, validation, testing sets with 0.6, 0.2, 0.2 splits yielding $14495, 4831, 4831$ samples, respectively. The LSTMs are trained with a learning rate of $1e-2$ until the training loss started to diverge. At this step, the learning rate was decreased to $1e-3$ and trained until validation accuracy did not improve further.

In the final step, the encoder, decoder, embedding generator and the LSTMs layers are combined. The entire network is trained with one image sequence per batch. Weights from the earlier training steps are used to initialise the network. The entire network is trained with label loss from the LSTM layers and segmentation loss from the decoder. Since the two losses were trained individually before this step, using the sum of two losses was adequate for the final step. ADAM optimiser with a learning rate of $1e-3$ was used and trained until the validation accuracy did not improve. The accuracy reported in all cases is on the test set.

Results

The right embeddings to LSTM layers influence their recognition capacity to a large extent. This is empirically evident from Table 5.7, where the differentiating factors that influence the accuracy are the embedding inputs to LSTMs. Cao et al. [12] argue adding homographic spatial transformer modules to VGG embeddings and homographic recurrent spatio-temporal transformer modules to C3D embeddings result in better embeddings to LSTMs, increasing their recognition accuracy. However, using SSAR network demonstrated that by using segmentation based embeddings we can get considerably better accuracy for the EgoGesture dataset. We posit that

Method	**Modality**	**Frames**	**Accuracy**
VGG16 + LSTM	RGB	Any	0.747
VGG16 + LSTM	RGB	40	0.808
VGG16 + RSTM (H) + LSTM	RGB	40	0.838
C3D + RSTTM (H) + LSTM	RGB	40	0.893
C3D + RSTTM (H) + LSTM	Depth	40	0.906
C3D + RSTTM (H) + LSTM	RGB + Depth	40	0.922
Segmentation based Embedding + LSTM (ours)	RGB	Any	**0.969**

Table 5.7: Comparison of results on Ego Gesture dataset to the state of the art. The results reported from Cao et al. [12] are in purple. Our network (results reported in green) produces considerably better accuracy on just RGB data during inference and can use all the images in a sequence, while the state of the art is limited to 40 frames per sequence and needs both RGB and Depth data for best results.

this is possible since any feature that does not belong to ego-hands is ignored by design in our network.

This point is further fortified by visualising embeddings created by the SSAR network are in Figure 5.13, as it can be seen that features on and around ego-hands contributed most for the Gradient-Class Activation Map (Grad-CAM images)[119]. Grad-CAM highlights the areas which contribute the most towards the final decision that is made by the network, in the case of SSAR network, it can be seen that areas belonging to ego-hands are highlighted the most. For gestures like Number6 fingers on ego-hands become essential for recognition. The SSAR network, as seen in Figure 5.13 correctly paid most attention to fingers in the RBG images. There is a need for understanding the spatial position of both the hands to discern the gesture, in the case of two-handed gestures. The activation maps showed attention rightly being paid to both the hands.

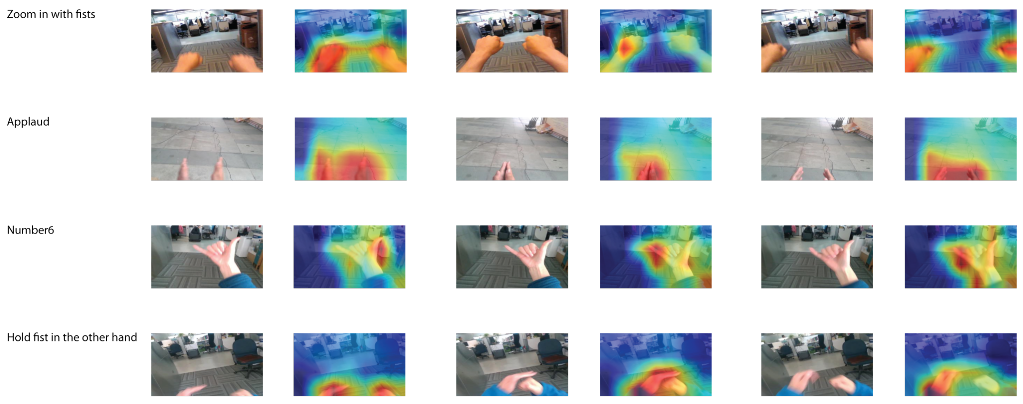


Figure 5.13: Gradient Class Activation Maps(Grad-CAM) [119] for showing the activations that correspond to the gesture. Each row has three RGB images and their corresponding Grad-CAM images from a sequence that belongs to a gesture. For illustration purposes, we choose gestures that use both hands and only one hand, to show the robustness of the network. The class activation maps rightly highlight the areas that belong to ego-hands, intuitively in some gestures as seen for Number6 gestures, the fingers on the hand contribute more towards identifying the gesture correctly.

Scenario	**State of the Art** [12]	**SSAR**
Walking	0.828	0.962
Stationary	0.866	0.972

Table 5.8: Accuracy on scenarios with (walking) and without (stationary) ego-motion. We outperform the state of the art in both scenarios.

The most confused gesture in our analysis using a confusion matrix (Figure 5.14) was the Sweep Cross. Out of the 65 test sample for this gesture, 57 were correctly classified, while 2 each were misclassified as Sweep Diagonal, Sweep Circle and Sweep Check-mark and 1 each as Beckon and Move Fingers upward(see Figure 5.15 for gesture illustrations). Since these gestures look quite similar, it is very probable for them to confused with each other.

The gestures were grouped into two disjoint sets, one containing ego-motion(walking) and another containing no ego-motion(stationary), following the same procedure from the state of the art. The results reported in Table 5.8 show that using segmentation based

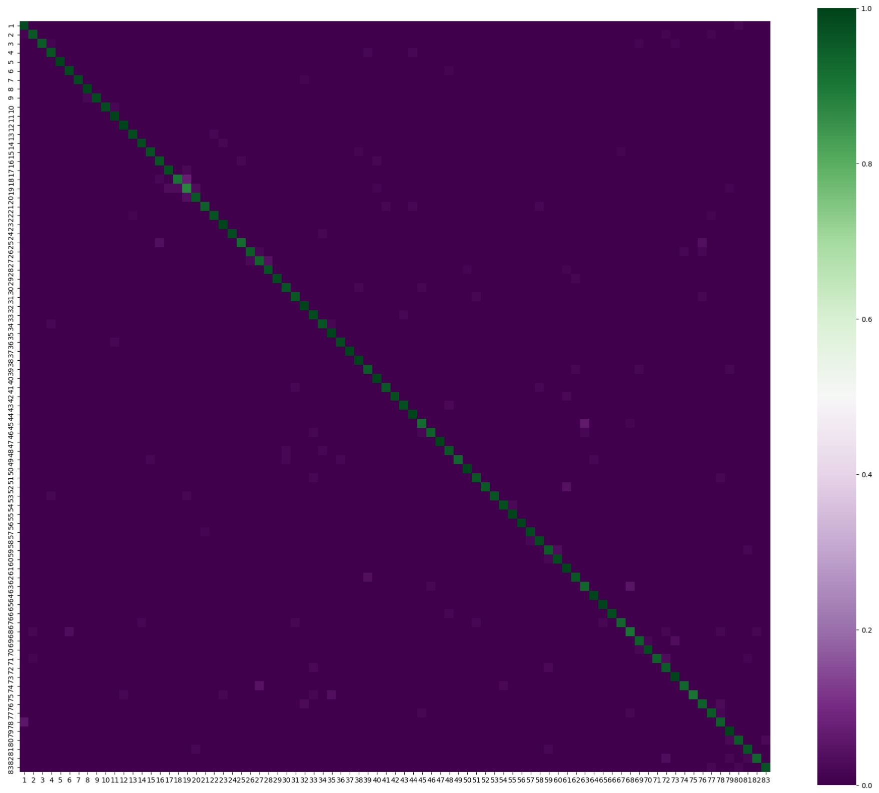


Figure 5.14: Confusion matrix for all the gestures.

embeddings is sufficient to compensate for ego-motion. SSAR network performed better in both the stationary and walking scenarios. The difference in accuracy between stationary and walking scenarios in our case is 1%, whereas it is 3.8% in the case of the state of the art, which further illustrates that our network by not paying attention to the context around ego-hands can deal with ego-motion better. The network proposed by Cao et al. [12] calculates and corrects for ego-motion through homography estimation. However, their estimation is not particularly conditioned on ego-motion, so any changes in the background can effect their homography estimation, which could be one of the causes for larger difference between the ego-motion vs no ego-motion scenarios when com-

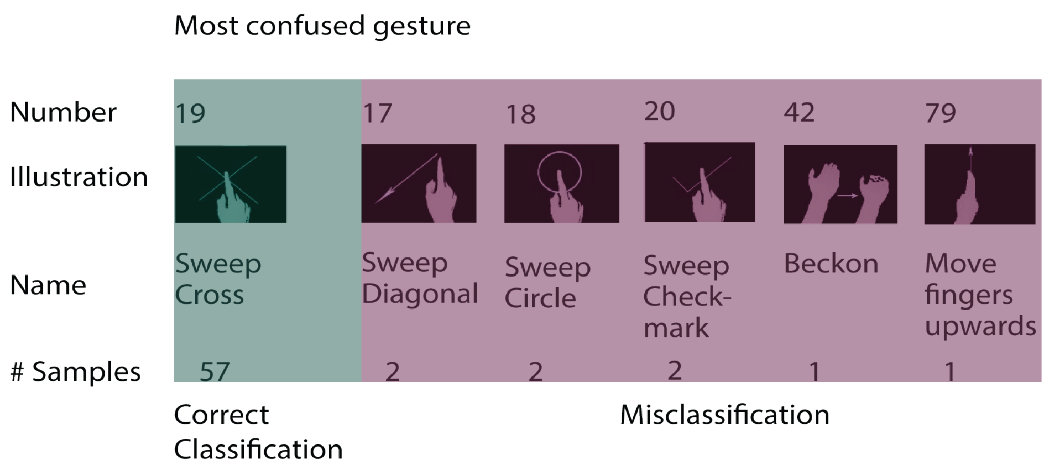


Figure 5.15: Confusion matrix for all the gestures. The most confused gesture as seen is Sweep Cross(19), out of the 65 samples 57 were correctly classified while 2 each were misclassified as Sweep Diagonal, Sweep Circle, Sweep Check-mark and 1 each as Beckon and Move Fingers Upwards.

pared to the performance of SSAR.

Validation

Ablation studies were performed and compared to the results reported by Cao et al. [12] to validate the usage of segmentation based embeddings. The decoder part of the SSAR network was ignored To remove explicit conditioning on ego-hands. The encoder was connected to embedding generator, which was further connected to LSTM. This network without decoder is trained end-to-end to recognise ego-hand gestures with cross-entropy loss. The backpropagation was performed similarly to other training until validation accuracy did not improve. The recognition accuracy from this simple embeddings + LSTM network (75.4%) was very close to VGG16+LSTM (74.7%) as reported in Table 5.9. To further understand the effect of using segmentation based embeddings the results of recognition accuracy on SSAR network after performing phase one and two of training as described in Section 5.3.3 are tested and reported. After training the encoder and decoder and creating segmentation based embeddings, LSTM were trained with these. The final fine-

Method	Accuracy
VGG16 + LSTM [12]	0.747
Simple Embedding + LSTM	0.754
Segmentation based Embedding + LSTM	0.947

Table 5.9: Table validating the use of segmentation based embedding. We use the encoder as described in section 5.3.3 to generate simple embeddings for the LSTM to recognise an ego gesture. In comparison to VGG16 + LSTM, this does not result in a significantly increased accuracy. The segmentation based embedding + LSTM approach, performs only the first part and second part of the training described in section 5.3.3. We forgo the final step of training to isolate the effect of using segmentation based embedding. The accuracy increases, when compared to the simple embeddings, which validates the use of segmentation based embeddings.

tuning step mentioned Section 5.3.3 was not performed to measure the impact of using segmentation based embeddings. The recognition accuracy improved considerably compared to the above two networks, thus validating the usage of segmentation based embeddings. It can be posited that these embeddings carry information particular to ego-hands, which lead to better generalisation capability than compared to simple embeddings as evident with the improved accuracy. The accuracy of the three networks is reported in Table 5.9.

5.5 Summary

Thus far, the concept of simultaneous segmentation and recognition(SSAR) was defined. Unlike the traditional approaches which calculated features on ego-hands after segmenting the images, SSAR calculates embeddings that can help spatially recognise the position of ego-hand in an image and simultaneously encode the gesture in an image. Based on the SSAR concept, a deep neural network architecture called the SSAR network was presented. This network was trained and tested for ego-hand gesture recognition on trimmed videos on three publicly available ego-hand gesture datasets including the Green Screen dataset introduced in Chapter 4.

On the AirGest dataset we achieved recognition accuracy of 96.25% compared to the then state of the art which was 93.75%, using SSAR network architecture. The size of our network was 20 times smaller (Table 5.3) than the then state of the art network, proving the efficacy of SSAR network empirically. On the EgoGesture Dataset we achieved a recognition accuracy of 96.9% compared to the state of the art which had a recognition accuracy of 92.2% (Table 5.5). On the Green Screen Dataset we achieved a recognition accuracy of 71.42% setting the benchmark for recognition accuracy. There is a considerable scope for improvement of recognition accuracy on the Green Screen Dataset compared to AirGest and EgoGesture dataset. This can be attributed to the fact that Green Screen dataset has a lot more realistic and hard scenarios like extreme motion blur, clipped fingers and large variations in gesture style which are absent in the other two datasets.

Thorough ablation and evaluation studies showing the efficacy of SSAR network were presented. However, for any ego-hand gesture recognition to be useful, it should work on untrimmed videos. Untrimmed videos contain ego-hand gestures information interspersed with other activity or non-activity, and the network presented in this chapter can not handle such scenarios. In the following chapter, the work recognising ego-hand gestures from untrimmed video is presented.

Chapter 6

Ego-hand Gesture Recognition on UnTrimmed Video

In this chapter, the work on recognising ego-hand gestures from untrimmed video is presented. Research on recognising gestures from trimmed videos is an interesting problem to solve, however owing to the assumption that the input contains images with only ego-hands performing a gesture it is not feasible to use in real scenarios. As it can be seen from Figure 6.1, real-world scenario consists of untrimmed video where ego-hand gesture images are interspersed with non-ego-hand gesture images. Previous works [9, 97] discussed in Chapter 3 used heuristics to train networks to solve this problem. Instead, network architecture with State Activation Gate(StAG) LSTM, intra-gesture(IG) loss is proposed and explained in the following sections. Also, shortcomings of Jaccard Index(JI) metric, a metric used to measure the performance of sequence to sequence classification problems are discussed and an improved metric Continuity Favouring Jaccard Index(CFJI) is presented. The network architecture with StAG, training strategy with IG loss and evaluation with the proposed CFJI metric is discussed in the rest of the chapter.

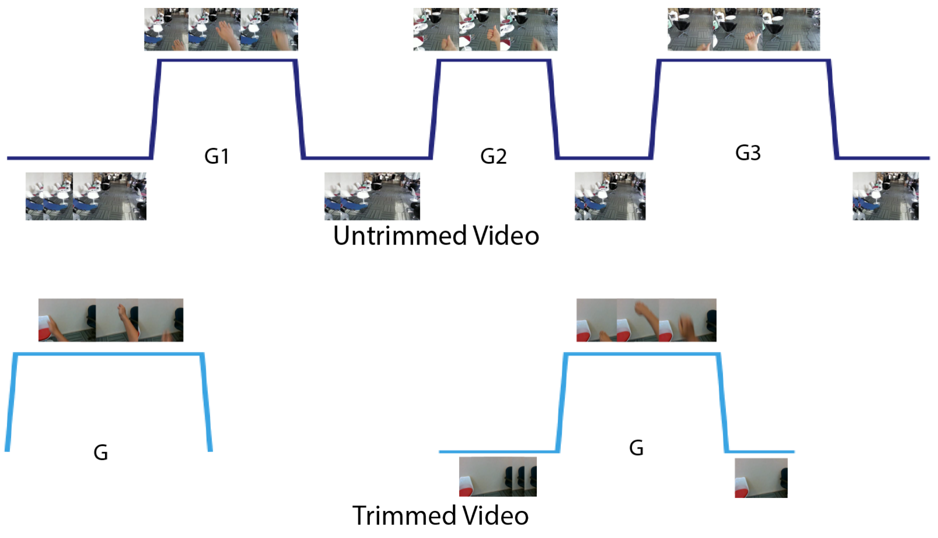


Figure 6.1: Top Row: Gestures in untrimmed video at different time steps, interlaced with non-gesture frames. Bottom Row: Usual ego-hand gesture recognition problems assume only one gesture per video with or without non-gesture frames.

6.1 Architecture

A combination of 3D convolutional neural network to encode spatio-temporal features and recurrent neural network to recognise ego-hand gestures from this sequence of encoded spatio-temporal features were used to design the network. We were inspired by various network architectures proposed in the past for gesture and action recognition like [74, 12, 96, 120] and SSAR presented in Chapter 5.1. In the following sections, the 3D CNN used for the visual encoder and StAG LSTM proposed for the sequence decoder are explained in detail.

There had been several 3D convolutional neural network architectures introduced like C3D, P3D, R(2+1)D [121, 122, 123]. The primary idea behind 3D convolutions is to capture both spatial features in an image and motion features across a given set of images. The input to a 3D convolutional neural network is usually a small sequence of consecutive images. R(2+1)D networks introduced by Tran et al. [123] were used for visual encoder part of the network. R(2+1)D is a variation of P3D network, which itself is an extension of ResNet

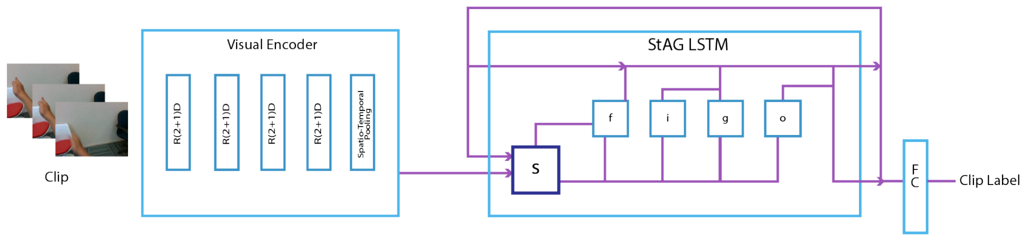


Figure 6.2: Architecture of our network. We used R(2+1)D Convolutions [24] for visual encoder and proposed StAG LSTM for sequence decoder.StAG LSTM has state activation gate that modulates the input and hidden state according to presence or absence of a gesture. StAG LSTM isexplained in Section Architecture and Training Overview, and illustrated with more details in Figure 6.3.

[124] from 2D to 3D space. Unlike P3D, R(2+1)D model uses a single type of block throughout the network and factorises 3D convolutions into 2D and 1D convolutions instead of bottlenecks. R(2+1)D networks were empirically shown to be superior to other 3D network architectures even with considerably less number of layers [123]. We choose R(2+1)D networks for the above reasons. The number of blocks in each layer was set to two, using the shallowest version of R(2+1)D Network. Though 3D convolutions can capture motion features across images, their limitation arises since they work across the current input and do not work beyond it. The implication of this limitation is to feed 3D Convolutional neural networks with larger clip size (consecutive images in video) as input if they had to adapt for longer input sequences. However, larger clip size places severe GPU memory requirements for training and inference. Recurrent neural networks are used in tandem with 3D CNN for recognition on large sequences of data to capture long-term dependencies in data.

6.1.1 StAG LSTM

A recurrent neural network is appended to the last encoding layer of the visual encoder to discern the temporal relations between video clips to give the network a larger temporal view of the video.

Basic recurrent neural networks have two major practical issues for training, exploding and vanishing gradients. Long-Short Term Memory or LSTMs as they are called in short were proposed by [46] particularly to deal with these two problems and since have been extensively researched and applied to action, and gesture recognition since then [74, 125, 102, 73, 12, 79]. To recap, LSTMs are made of input, forget, output gates represented by equations 6.1a ... 6.1d. These gates influence the current cell and hidden states deciding what information needs to be remembered, forgotten and to what extent. The hidden state from each unit is then connected to a fully connected layer filter to generate a gesture label using Softmax classification. The full architecture of the network is illustrated in Figure 6.2.

$$i_t = \sigma(W_{xi}x^t + b_{xi} + W_{hi}h^{t-1} + b_{hi}) \tag{6.1a}$$

$$f_t = \sigma(W_{xf}x^t + b_{xf} + W_{hf}h^{t-1} + b_{hf}) \tag{6.1b}$$

$$g_t = \tanh(W_{xg}x^t + b_{xg} + W_{hg}h^{t-1} + b_{hg}) \tag{6.1c}$$

$$o_t = \sigma(W_{xo}x^t + b_{xo} + W_{ho}h^{t-1} + b_{ho}) \tag{6.1d}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{6.1e}$$

$$h_t = o_t \odot \tanh(c_t) \tag{6.1f}$$

However, in the case of ego gesture recognition on untrimmed videos, the number of frames with non-gesture labels is higher compared to frames with one particular gesture. In such a case using the entire video for training leads the network to overfit on non-gesture data if plain LSTMs are used, as reported in the experiments and analysis section 6.4. Heuristics like selecting a part of the video for

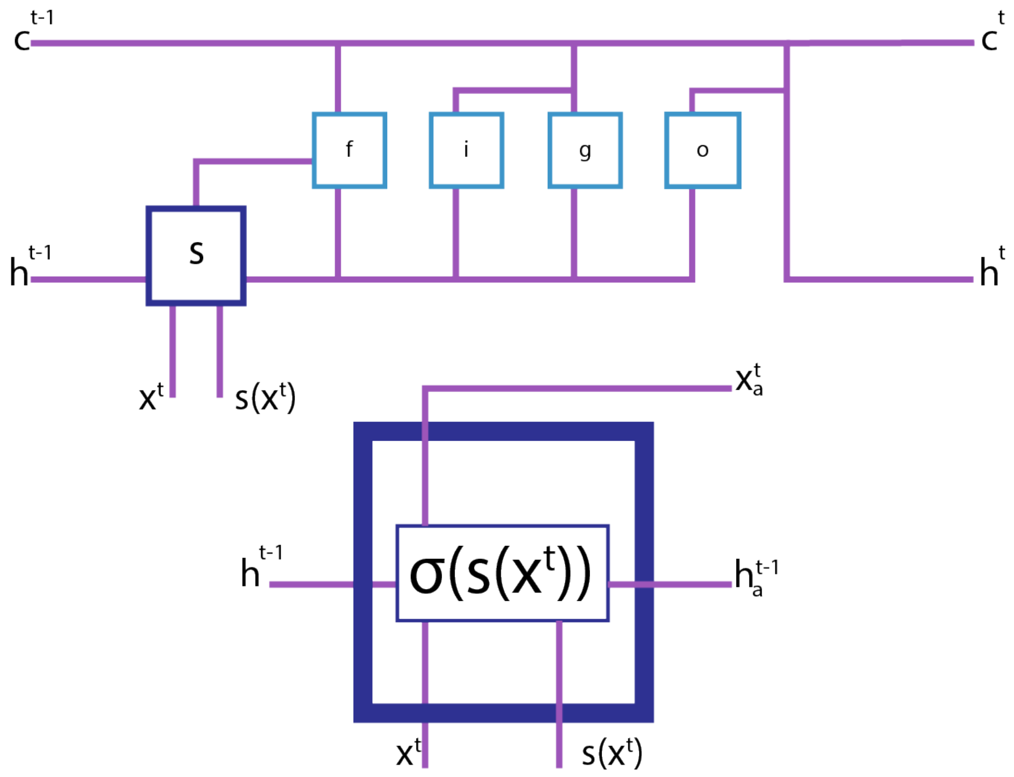


Figure 6.3: Stag LSTM. State Activation Gate is added to standard LSTM to modulate the flow of input and hidden state. In addition to hidden state and cell state StAG LSTM also outputs the activation mapping for input to optimise the activation function s. For brevity only the state activation gate is expanded.

training the recurrent network [9, 97] and using a particular amount of frames without gestures before and after the gesture is performed [9] were used to deal with the issue of overfitting. We introduced an additional gate to the LSTM framework in order to avoid usage of such heuristics. We call this state activation gate, the intent of a state activation gate is to modulate the activation of input and hidden state depending on the current state of the input. The equations 6.2a, 6.2b and 6.2c represent the state activation gate and the gate being applied to the input and the hidden state. Figure 6.3 illustrates the addition of State Activation Gate to LSTM. As it can be seen that the new activated input, hidden state are used for the rest of the LSTM instead of the original states (6.2d . . . 6.2i).

$$s_t = \sigma(s(x^t)) \tag{6.2a}$$

$$x_a^t = x^t \odot s_t \tag{6.2b}$$

$$h_a^{t-1} = h^{t-1} \odot s_t \tag{6.2c}$$

$$i_t = \sigma(W_{xi}x_a^t + b_{xi} + W_{hi}h_a^{t-1} + b_{hi}) \tag{6.2d}$$

$$f_t = \sigma(W_{xf}x_a^t + b_{xf} + W_{hf}h_a^{t-1} + b_{hf}) \tag{6.2e}$$

$$g_t = \tanh(W_{xg}x_a^t + b_{xg} + W_{hg}h_a^{t-1} + b_{hg}) \tag{6.2f}$$

$$o_t = \sigma(W_{xo}x_a^t + b_{xo} + W_{ho}h_a^{t-1} + b_{ho}) \tag{6.2g}$$

$$c_t = f_t \odot c_{t-1} + i_t \odot g_t \tag{6.2h}$$

$$h_t = o_t \odot \tanh(c_t) \tag{6.2i}$$

The choice of function s for state activation gate is left to the user, the intent of this function is to map the input x_t at time t to a value between 0 and 1, such that the gate can selectively modulate the input and hidden state that needs to enter the next gates in LSTM. For the purpose of ego-hand gesture recognition, s was chosen to be a fully connected neural network that infers if the current frame x_t has a gesture or not. In the case of EgoGesture dataset a fully connected neural network with one layer was enough to discern the presence or absence of a gesture. The scenario is simple because, the presence of an ego-hand implied a gesture being performed. In more complex cases where an ego hand could visible but is not performing a gesture more complex s can be chosen. The output of the function s is used in binary cross entropy loss function to learn its parameters, in addition to the loss function back propagated from the hidden state. This allows the network to actively modulate the flow of input and hidden state depending on the presence or absence of gesture in the current frame, allowing for the parameters in the LSTM to be learnt for gestures instead of the no gesture frames during training. There is no need to modulate the cell state separately, as evident from equation 6.2h, since inputs to this gate are already activated input and hidden states. The following section details the training and evaluation procedure. Tables 6.1, 6.2 summarise all the hyperparameters that are used to design the visual encoder and StAG LSTM.

	Blocks	**Input Channels**	**Output Channels**
STC	1	c	64
STR	2	64	64
STR	2	64	128
STR	2	128	256
STR	2	256	d

Table 6.1: Visual Encoder Network Parameters (STC: Spatio Temporal Convolution) and (STR: Spatio Temporal ResNet). The structure of STR, STC is further explained in Figure 6.4. c depends on the input modality $1, 3, 4$ for $depth, rgb, rgbd$ respectively. d depends on the complexity of the dataset, d is set to 512 for the EgoGesture Dataset and to 256 for the NVIDIA gesture dataset.

	input	**Hidden**
EgoGesture [9]	512	128
NVIDIA Gesture [73]	256	52

Table 6.2: StAG LSTM Network Parameters. The size of cell state is same as the hidden state. The input size depends on the output of the Spatio Temporal Pooling layer in the visual encoder. The hidden size is set depending on the number of gestures to recognise.

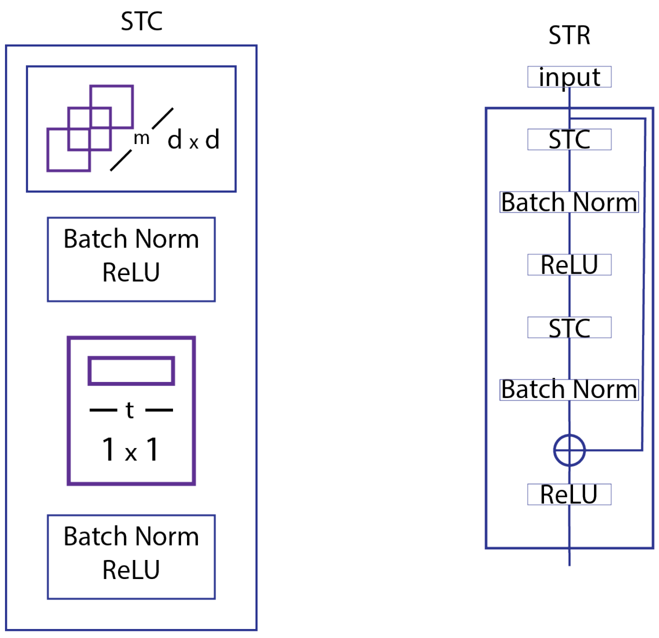
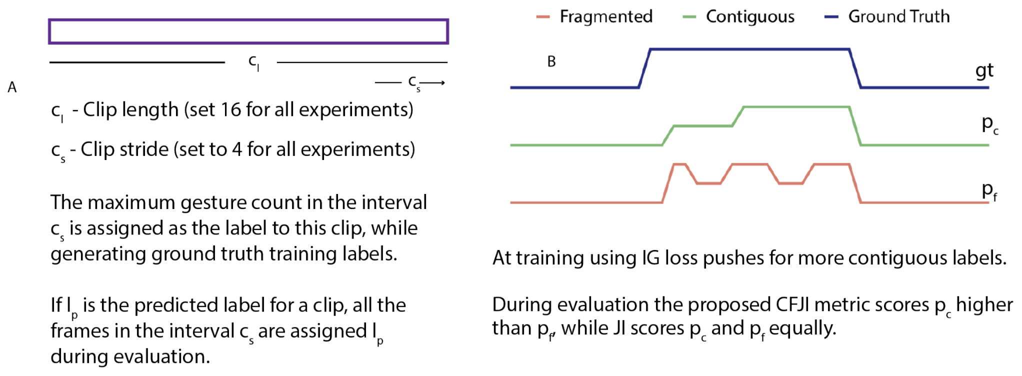


Figure 6.4: The spatio-temporal convolution(STC) block from R(2+1)D network [123]. They factor the 3D convolution into a 2D convolution, followed by batch normalization and relu and 1D convolution followed by batch normalization and relu. The spatio-temporal resnet(STR) blocks consist of the standard ResNet block with 3D convolution replaced by the proposed STC block.

6.2 Training



c_l - Clip length (set 16 for all experiments)

c_s - Clip stride (set to 4 for all experiments)

The maximum gesture count in the interval c_s is assigned as the label to this clip, while generating ground truth training labels.

If l_p is the predicted label for a clip, all the frames in the interval c_s are assigned l_p during evaluation.

At training using IG loss pushes for more contiguous labels.

During evaluation the proposed CFJI metric scores p_c higher than p_f, while JI scores p_c and p_f equally.

Figure 6.5: (A) Figure illustrating our label assignment procedure during training and evaluation.(B) Figure showing how CFJI metric would score sequences compared to JI metric.

The problem of ego-hand gesture recognition in untrimmed video is briefly formalised, before looking into the training methodology. V is a video containing $x_1 \ldots x_t$ frames, $l_1 \ldots l_t$ are the corresponding gesture labels, where $0 \leq l_i \leq n$, n being the number of gestures, 0 label corresponds to no gesture. We need to find a function f that maps $x_1 \ldots x_t$ to $l_1 \ldots l_t$. In the current case f is the combination R(2+1)D and StAG LSTM.

The video is chunked into clips of length c_l frames with a stride c_s.The following strategy is used to assign a label to the clip: if a clip c is denoted by frames $[x_t, x_{t+c_l-1}]$ and $[l_t, l_{t+c_l-1}]$ are their corresponding labels. As illustrated in Figure 6.5 a the clip label is set to maximum count of gesture labels in the interval $[l_{t+c_l-s}, l_{t+c_l-1}]$. c_l was set to 16 and c_s to 4 for all the experiments performed.

Cross entropy loss is used with the number of classes set to $n+1$, where n is the total number of gestures, and the extra label set to 0 for non-gesture frames to train the visual encoder. It should be noted that unlike training methodology in [9], which uses only some frames from videos, we used all the clip regardless of the presence or absence of a gesture. Once the visual encoder is trained, the output from the final spatio-temporal pooling layer is stored for each video clip-wise sequentially.

The full length of the encoded video is used as input to StAG LSTM, and we used all the videos in training split in a single batch. It must be noted that the full video is not necessary for the inference stage; it is only needed for training. StAG LSTM has two outputs, one from state activation function described in equation 2a and hidden state. Binary cross entropy loss is used for the state activation function, and the hidden state is connected to a linear layer with $n+1$ neuron, n being the number of gestures. The loss function we used for training StAG LSTM is represented by the equation 6.3.

$$L = L_{StA} + L_C \tag{6.3a}$$

$$L_C = L_{CE} + L_{IG} \tag{6.3b}$$

where L_{StA} is the state activation loss, as described earlier, binary cross entropy is used to represent this loss. And L_{CE} is the standard cross entropy loss, and L_{IG} is the intra-gesture loss which is explained in the section below.

6.2.1 Intra-Gesture Loss

A subject performs the same gesture from the beginning to the end, so the gesture labels also remain the same throughout the gesture(illustrated in Fig6.6). However, this property is not utilised by methods like [97, 9] which adopt networks trained for trimmed gesture recognition to untrimmed gesture recognition. This can have considerable impact on the performance metrics which measure sequence similarity metrics, and this is very evident in the case of Kopuklu et at [97], further explored in Section6.4.1.

To encourage consecutive frames to have the same gesture labels Intra-gesture loss was introduced. Intra-gesture loss is defined by equation 6.4

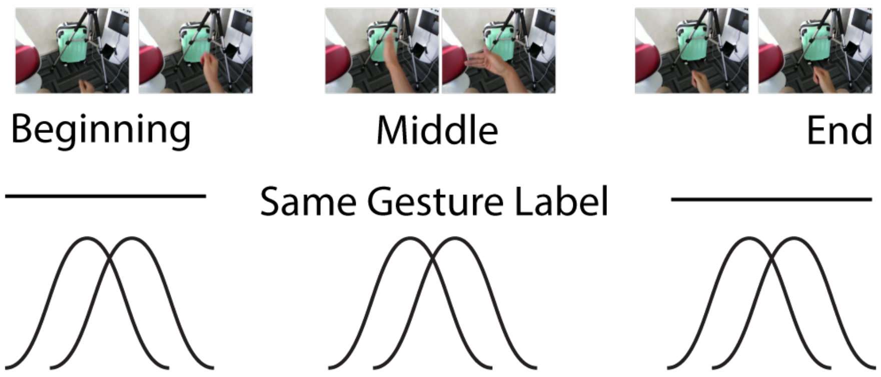


Figure 6.6: When a subject performs a gesture, the label is the same from the beginning to the end. This phenomena is the basis for Intra-Gesture(IG) Loss. The IG loss encourages the network to give same label to consecutive frames, by selectively minimising the Kullback-Liebler distance between two consecutive prediction probabilities.

$$L_{IG}(P||Q) = \sum_{x \in X} \delta_{xl} * P(x) * ln(P(x)/Q(x)) \tag{6.4}$$

where l is the ground truth label, δ is the Kronecker delta. Intra gesture loss is the product of Kronecker delta and Kullback-Liebler distance. We applied this loss to consecutive frames with gestures. The idea of intra-gesture loss is to penalise if the distribution of prediction for two consecutive frames is different. However, with the introduction of Kronecker delta, this penalty is selectively applied to the gesture label that matches the ground truth. In Section 6.4 the usefulness of this loss function is validated.

6.3 Evaluation

6.3.1 Intra-Gesture Loss

Jaccard Index has been used as an evaluation metric for gesture recognition on untrimmed videos [107, 9]. Jaccard index as indicated by equations $5a, 5b, 5c$, measures the relative overlap between

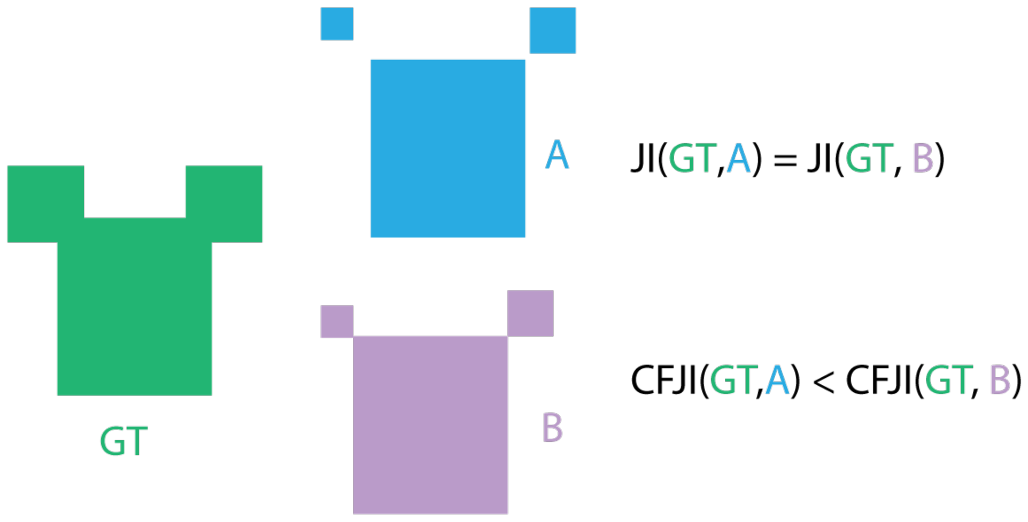


Figure 6.7: The current Jaccard Index(JI) metric scores the predictions A & B the same in comparison to the ground truth. However, the Continuity Favouring Jaccard Index(CFJI) metric scores B > A since prediction B is more continuous and less segmented compared to A. The CFJI metric evaluates the performance of a network prediction not only based on the intersection with ground truth, but also the continuity of the prediction.

the ground truth and predicted sequences.

$$JI(G||P)_{s,i} = \frac{G_{s,i} \bigcap P_{s,i}}{G_{s,i} \bigcup P_{s,i}} \tag{6.5a}$$

where $G_{s,i}$ is the ground truth part of sequence which has gesture i, similarly $P_{s,i}$ is the predicted part of sequence which has gesture i.

$$JI(G||P)_s = \frac{1}{N} \sum_{i=1}^{N} JI_{s,i} \tag{6.5b}$$

$$JI(G||P) = \frac{1}{S} \sum_{s=1}^{S} JI_s \tag{6.5c}$$

where $JI(G||P)_s$ measures the Jaccard Index between two sequences with N different gestures in it and $JI(G||P)$ is the mean Jaccard Index of all the sequences in the set with cardinality S. However, the Jaccard Index does not take the continuity of predicted labels into account. For example if {0, 0, 1, 1, 1, 1, 1, 1, 0, 0} are ground truth labels G for a sequence, {0, 0, 1, 0, 1, 0, 1, 0, 0, 0} is a set of predicted labels P_1, and {0, 0, 0, 0, 0, 1, 1, 1, 0, 0} is another set of predicted labels P_2, then $JI(G||P_1) = JI(G||P_2)$. A metric M that would score $M(G||P_2) > M(G||P_1)$ is more suitable, since it would be more useful if labels are continuous, instead of segmented.

6.3.2 Continuity Favouring Jaccard Index

A new metric Continuity Favouring Jaccard Index(CFJI)(see Figure 6.7) is defined that would favour continuity and also measures the relative overlap between ground truth and predicted labels. The following equations define CFJI.

$$CFJI(G||P)_{s,i} = \begin{cases} \frac{N_{s,i}(G)}{N_{s,i}(P)} * \frac{G_{s,i} \bigcap P_{s,i}}{G_{s,i} \bigcup P_{s,i}}, \\ \text{if } N_{s,i}(P) \geq N_{s,i}(G) \\ \\ \frac{N_{s,i}(P)}{N_{s,i}(G)} * \frac{G_{s,i} \bigcap P_{s,i}}{G_{s,i} \bigcup P_{s,i}}, \\ \text{if } N_{s,i}(G) > N_{s,i}(P) \\ \\ 0, \\ \text{if } N_{s,i}(G) = 0 \text{ or } N_{s,i}(P) = 0 \end{cases} \tag{6.6a}$$

where $N_{s,i}$ is the number of continuous segments of gesture i in sequence s. It can be seen that if the number of continuous segments for a gesture is different in predicted sequence labels compared to the ground truth our metric CFJI decreases the score, if they are the same it retains the JI value.

$$CFJI(G||P)_s = \frac{1}{N} \sum_{i=1}^{N} CFJI_{s,i} \tag{6.6b}$$

$$CFJI(G||P) = \frac{1}{S}\sum_{s=1}^{S} CFJI_s \tag{6.6c}$$

In the following section, we present detailed analysis, experiments and ablation studies performed on two publicly available ego-hand gesture datasets that empirically show the efficacy of our network architecture with StAG LSTM and training methodology.

6.4 Experiments and Analysis

We used EgoGesture [12], the largest ego-hand gesture dataset available publicly, and NVIDIA Gestures [73] to validate our network architecture and training with StAG LSTM.

6.4.1 Experiments on EgoGesture Dataset

EgoGesture dataset has a total of 2081 videos with gestures being performed at irregular intervals recorded in different environments with varied lighting conditions and has 83 different gestures performed by 50 subjects. This set of 2081 videos is split into 1239, 411, 431 videos of training, validation and testing set respectively and we followed the same settings mentioned in [9] for reporting the results.

After the visual encoder is trained, the feature vectors at spatio-temporal pooling layer (of length $512 \times n$ where n is the number of clips in the video) are stored. The hidden size in StAG LSTM is set to 128. The activation function is trained for first 40 epochs, by using only the activation function loss. The classifier loss is used from 41st epoch until the validation error does not improve any more. The output of the network is assigned per clip to the slide window (illustrated in Figure 6.5A) to generate full length of video labels to compare with the ground truth labels. The metrics scores are reported on the test set. Table 6.3 compares our network's performance with the state of the art.

Method	Modality	Heuristics	JI
C3D+STTM [12]	RGB	Yes	0.670
R(2+1)D + LSTM	RGB	No	N/A
C3D + LSTM	RGB	No	N/A
R(2+1)D + StAG LSTM(ours)	RGB	**No**	**0.684**
C3D+LSTM [9]	Depth	Yes	**0.710**
R(2+1)D + StAG LSTM(ours)	Depth	**No**	**0.710**
R(2+1)D + LSTM	Depth	No	N/A
C3D + LSTM	Depth	No	N/A
C3D + LSTM [9]	RGBD	Yes	0.718
R(2+1)D + LSTM	RGBD	No	N/A
C3D + LSTM	RGBD	No	N/A
R(2+1)D + StAG LSTM(ours)	RGBD	**No**	**0.722**

Table 6.3: Jaccard Index scores for various networks [12, 9]. Our performance on the JI metric is similar or better than the start-of-the-art network without using heuristics. Heuristics are employed by current methods to adapt networks trained on trimmed videos to untrimmed videos. Our network which uses **StAG LSTM (6.1.1) does not employ heuristics** and performs better or similar on various modalities in comparison to existing networks. For networks using simple LSTM without heuristics, the JI column is labelled as N/A. The networks with simple LSTMs trained without heuristics did not converge, this phenomena is further explained in Section6.4.1.

The scores for RGB modality in comparison to Depth and RGBD modality is less in both state of the art and our network. This score is attributed to the fact that RGB images have a lot of background signal which needs to be filtered out in addition to ego-hands. Depth images can be easily thresholded to extract ego-hands, but they lose some finer details, so they perform better than RGB. RGBD modality outperforms both RGB and depth modalities because it combines the information from these two modalities. The work by Kopuklu et al. [97] is the other network that recognises ego-gestures from untrimmed videos in EgoGestures dataset. However, the performance of the two-stream network proposed by them is deficient because the crux of their algorithm is a proposed postprocessing step. Their network scored 0.484 on JI metric without post-processing. Applying their postprocessing does not output a sequence label for making a direct comparison. Their network's performance relied on a ResNext [112] network which was used as the primary classifier. However, it can not form long term dependencies, and its performance is limited to the size of the clip, which was chosen to be 32 in their case. It must also be noted that [97] used training + validation set for training and then reported the scores on the testing set. The effects of StAG LSTM, the usage of Intra-Gesture loss for training their scores on JI and CFJI metrics are discussed in the following section.

Ablation Studies

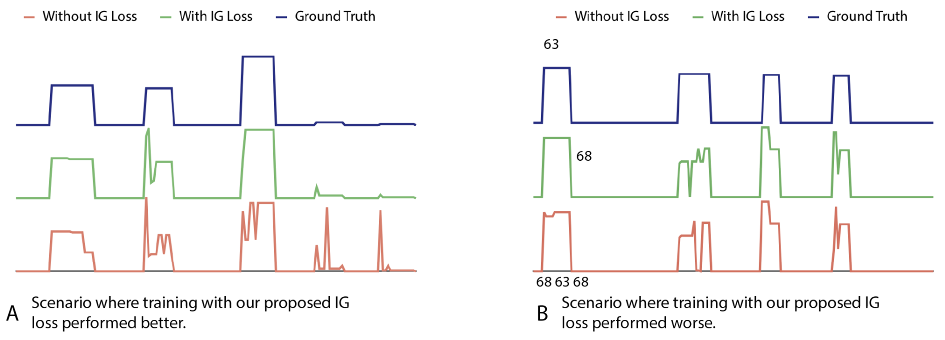


Figure 6.8: (A) Labels predicted for video Subject 14, Scene 3, Task4. The model trained with IG loss showed fewer variations in predictions compared to those trained without IG loss yielding better JI and CFJI scores. (B) Labels predicted for Subject 11, Scene 2, Task5, even though the number of variations is less, but the model trained with IG loss performed worse. The model trained with IG loss predicted the wrong label consistently during the gesture is performed, but the model trained without IG loss intermittently predicted correct labels.

One of the problems to solve in training deep neural networks for ego gesture recognition using untrimmed video is a large number of training images containing no gestures compared to that with a particular gesture. Methods like [9, 97] use heuristics like carefully choosing a part of the training sequence, and using weighted cross entropy loss(weights chosen are another set of heuristics) to deal with this issue.

We trained a plain LSTM with the same procedure used to train

StAG LSTM. The training loss stopped decreasing much earlier compared to StAG LSTM. The metrics on the validation set also followed the same trend. Figure 6.9 shows the training loss and validation metrics for both LSTM and StAG LSTM. It can be seen that using StAG LSTM leads to better training loss and validation accuracy while eliminating the need for heuristics that are used in [9, 97]

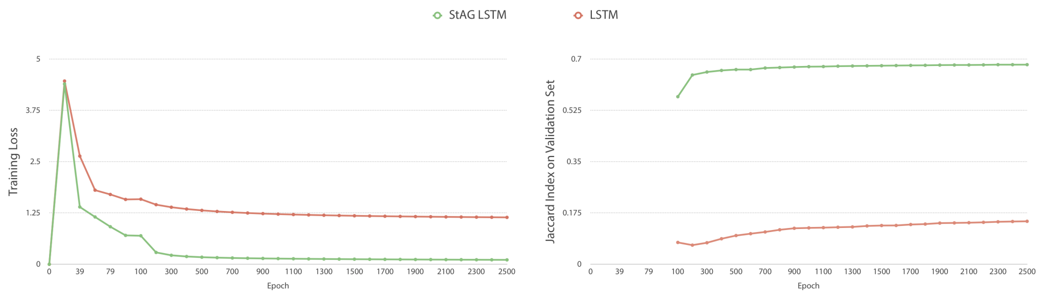


Figure 6.9: Training Loss for LSTM vs StAG LSTM. We can see that training loss stops decreasing for LSTM. However, it continues to improve for StAG LSTM. This improvement is also reflected in the validation Jaccard Index scores.

Loss	Modality	JI	CFJI
$L_{StA} + L_{CE}$	RGBD	0.718	0.676
$L_{StA} + L_{CE} + L_{IG}$	RGBD	0.722	0.681
$L_{StA} + L_{CE}$	Depth	0.706	0.664
$L_{StA} + L_{CE} + L_{IG}$	Depth	0.710	0.665
$L_{StA} + L_{CE}$	RGB	0.682	0.639
$L_{StA} + L_{CE} + L_{IG}$	RGB	0.684	0.642

Table 6.4: Jaccard Index and Continuity Favouring Jaccard Index metrics for training with and without Intra Gesture loss. Adding the Intra Gesture loss component improves both the metrics across modalities.

The network is trained with and without IG loss with the three modalities to compare the effectiveness of the loss function. It could be observed that across all the three modalities using Intra-Gesture loss improves both the Jaccard Index and the Continuity Favouring Jaccard Index as reported in Table 6.4. In Figure 6.8 two results are reported, one where using IG loss helps get better CFJI score and the other where it fails to. Gesture 63 (Thumbs Upwards) is confused for gesture 68 (Thumbs Forward), in the case that does not employ IG loss we can see that part of the gesture is labelled as 63, but when IG loss is used the entire gesture is labelled 68 (incorrectly). The incorrect labelling could be a potential downside, meaning if a gesture gets mislabeled in the beginning, it could retain the wrong label until the gesture ends. However, this behaviour occurs less in comparison to predicting the correct labels, hence the overall improvement in both JI and CGJI metrics. The increase in CFJI metric is relatively more compared to JI metric across modalities when we used Intra-Gesture loss. This increase showed us that using Intra-Gesture loss helps decrease fragmentation. In this scenario, CFJI metric helped us understand the continuity of predicted labels while maintaining intersection over union.

Non Gesture Suppression

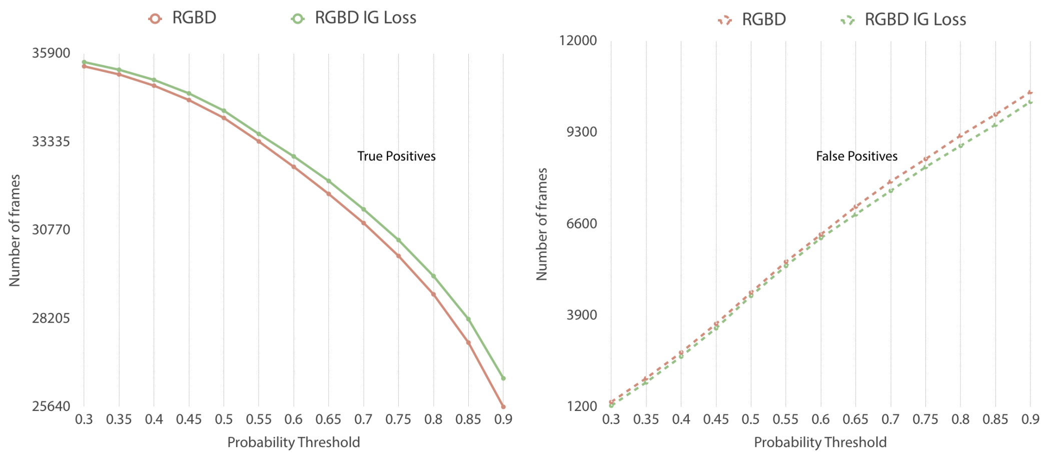


Figure 6.10: Non-Gesture Suppression: The output of our network is a set of probability values assigned to each gesture known. The current frame is labelled with the gesture that has maximum value amongst this set. We can set a threshold value τ so that only labels with values $> \tau$ are considered gesture frames, and everything else is considered a non-gesture frame. The higher the value of τ, the lesser false positives are labelled, but this comes at the cost of losing some true positives. However, using IG loss with cross entropy loss, we can set τ to higher value compared to using just cross entropy loss. As shown in this figure, with IG loss the number of false positives decrease (green dotted lines) yet, the number of true positives increases(solid green line) in comparison to a network trained without IG loss(orange lines).

One of the problems to solve in continuous gesture recognition is to identify non-gestures. Non-gestures are movements of hands seen by the camera but are not part of the set of gestures to be identified. Identifying non-gestures helps the VR/AR systems to stop taking a requisite action under the wrong assumption that the movement of hands of the user is an interaction. One of the ways to achieve non-gesture suppression is to collect data specific to non-gestures consisting of random hand movements and making the network recognise a gesture from non-gesture. However, this involves extra data collection, which is not a part of the EgoGesture dataset. Another solution to solve this problem is to use a higher threshold for the probability values of the gesture labels which are the output of the Softmax layer of the network (Section 6.1).

As seen in Figure 6.10, setting the threshold too high leads to suppression of true positives, decreasing the accuracy of the network. However training the network with IG (Section 6.4) and CE loss leads to better accuracy when compared to a network trained with only CE loss, the accuracy difference increased with an increasing threshold. This behaviour could be attributed to the fact that IG loss when used to train StAG LSTMs, rewards if the consecutive frames have same gesture label, but penalises the network when they are different enabling the network to learn the correct contiguous label. As illustrated, a higher threshold can be used to suppress non-gesture frames when trained with IG and CE loss together.

Early Detection Analysis

Detecting a gesture early can help making interactions with virtual elements smoother. Each gesture in the test set is divided into four equal temporal segments, corresponding to 0-25%,25-50%, 50-75%, 75-100% of the duration of the gesture, to analyse early gesture detection performance of training our network with and without IG loss. The relative difference in mean average precision(mAP) in each bucket is plotted (Figure 6.11), it can be seen that most of the gain in mAP scores using IG loss occurred in the 25-50% and the 0-25% temporal segments of a gesture performance. The gain in the

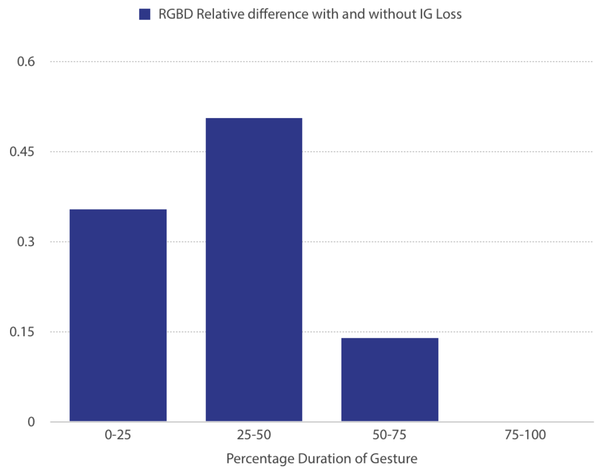


Figure 6.11: Early detection with IG loss: The length of each gesture is divided into four buckets: (0-25, 25-50, 50-75, 75-100)%, this graph plots the relative mean average precision of network trained with IG loss versus without IG loss for each bucket. The most gains of the network trained with IG loss can be seen in the 25-50% bucket, and this enables us to perform early detection of the gesture which in turn can make interactions with virtual elements quicker.

first two temporal segments could be attributed to the fact that using IG loss forces the network score sequential frames with the same gesture label leading to early detection mAP gains. As the gesture performance nears the end, the confidence in labelling the gesture increases, hence the relative gains are lower when compared to the performance at the beginning.

6.4.2 Experiments on NVIDIA Gesture Dataset

NVIDIA Gesture dataset [73] was introduced before the EgoGesture dataset. The dataset contains gestures which can be used in user interface scenarios in untrimmed videos, making it a suitable candidate for testing our network and training approach. It has 1050 videos in the training set and 482 videos in the test set (there is no validation split).

The number of gestures in this dataset is 25. We scaled down the model accordingly, changed the encoding feature layer size in R(2+1)D to 256, and hidden size in StAG LSTM to 52. The batch size for training visual encoder is set to 11 and used the entire batch for training StAG LSTM. The detection ROC+AUC scores are reported in Table 6.5.

Method	**Modality**	**ROC AUC**
3D-CNN + CTC	Depth	0.91
R(2+1)D + StaG LSTM	Depth	**0.92**

Table 6.5: ROC+AUC Detection Scores on NVIDIA Gestures dataset.

6.5 Summary

In this chapter, the need for recognising ego gestures from untrimmed video was discussed. We have empirically shown why the existing LSTMs does not train well without the help of heuristics which were employed in earlier methods. An additional gate called the State Activation Gate (StAG) was introduced to the LSTM framework to avoid some of the heuristics employed. A deep neural network architecture employing StAG LSTM was proposed. A novel intra-gesture loss function that encourages consecutive image frames to have the same gesture label was introduced along with a new evaluation metric(CFJI) that favours continuous gesture labelling. The deep neural network with StAG LSTM trained using the proposed IG loss function outperformed the current state of the art neural network for recognising ego-hand gestures from untrimmed video whilst employing a lesser number of heuristics compared to the existing methods. The proposed CFJI metric was shown to be helpful in objectively evaluating networks to label images with the same gesture consistently across which was not possible with Jaccard Index metric, one of the standard metrics used to measure the performance of untrimmed video recognition networks.

Chapter 7

Conclusions and Future Work

This chapter concludes the research presented in the thesis by summarising the contributions done to achieve the objectives for ego-hand gesture recognition conceptualised in the beginning. We also present the collaborative work with NVidia that is currently happening as an extension of the research presented and possible future extensions that can realise the idea of using ego-hand gestures as natural interfaces for AR/VR in the Future Work section.

7.1 Summary

As stated earlier, motivated by the idea of using ego-hand gestures as natural interfaces to interact with virtual environments in head-mounted AR and VR devices, the objectives for research in this thesis were laid out to be the following.

1. Define characteristics of a good ego-hand gesture dataset that can be used to train ego-hand gesture recognition deep neural networks. Create an ego-hand gesture dataset that addresses the problem of adding new gestures with lesser effort compared to the existing publicly available datasets while adhering to the characteristics defined.

2. Design a deep neural network architecture that focuses on ego-hands in the images to improve the state of the art recognition benchmarks on existing datasets and the new dataset on trimmed videos.
3. Identify the problems in using the existing ego-hand gesture recognition networks for real-world scenarios, devise solutions to solve some of the identified problems to move towards more usable neural networks.

Green Screen Dataset, a new ego-hand gesture dataset was introduced to reduce the laborious data collection process to satisfy the first objective. The details of the data collection process using a green screen as a background and how the green screen data could be augmented to train deep neural networks to recognise ego-hand gestures was outlined concisely in Chapter 4. A thorough comparison of the number of steps needs to add a new gesture to the Green Screen dataset vs EgoGesture, and AirGestAR datasets were analysed. We showed that using the Green Screen dataset and process outlined for creating augmented training data, ego-gesture recognition networks can be effectively trained in Chapter 5.

A novel deep neural network architecture that pays attention to ego-hands in a picture to recognise the gesture being performed was presented in Chapter 5. This network operates on the premise that encoding the spatial position of ego-hand in an embedding would improve gesture recognition accuracy. To realise this idea of simultaneously encoding an ego-hand and recognising the gesture performed, a network architecture consisting of encoder-decoder, embedding generator and sequence labeller was presented. This network was trained and tested on three different ego-hand gesture datasets, including the Green Screen dataset. In all the three settings, the proposed network advanced the state of the art accuracy metrics. We have shown that using ego-hand based embeddings can lead to better performance even with much ego-motion in the video compared to networks that estimate the ego-motion and compensate for the estimated motion. We empirically verified that paying specific attention to ego-hands decreases the need for a separate ego-motion estimation and compensation step. Ablation studies performed to validate the efficacy of the ego-hand segmentation based embeddings showed that these embeddings act as a regulariser, helping the network achieve the state of the art performance.

A novel approach to recognising ego-hand gestures from untrimmed video was proposed to accomplish the final and third objective. The existing approaches mostly solved the problem of recognising ego-hand gestures from trimmed video. However, to aid interaction with virtual objects in AR and VR environments, ego-hand gesture recognition needs to work on untrimmed videos. Untrimmed videos contain many images without any ego-hand gestures in them. The recognition network needs to identify these images in the video to ignore and pay attention to only the images containing gestures. The existing methods used heuristics and adapted the networks trained for ego-hand gesture recognition on trimmed videos to address recognition on untrimmed videos. Our proposed StAG LSTM, an extension to LSTM framework overcomes the need for using heuristics. The network architecture proposed works directly for training using untrimmed videos without using any heuristics. We also proposed a new loss function(IG Loss) and a better evaluation metric(CFJI metric). The proposed loss function helps early recognition of gestures giving the AR/VR system more time to react to the user's gesture and decrease the system response time. The StAG LSTM network with IG loss was extensively tested and evaluated on EgoGesture dataset beating the state of the art performance metrics.

7.2 Future Work

Ego-hand gesture recognition can also be used for communication in co-working environments with robots. We are currently exploring this idea of using ego-hand gestures as intent predictors in work environment shared with robots. The research done to explore this idea is presented in the next section, while we present possible extensions to ego-gesture recognition in the latter section.

7.2.1 Intent Prediction for Co-working with Robots Through Ego-hand Gesture Recognition.

The deployment of robots, both stationary and mobile, have been increasing recently in all industrial sectors to keep up with the production demands. While stationary robots can be used on manufacturing lines to assist with various tasks, mobile robots can transport and move objects helping humans on the floor and alleviate their physical labour. In both these scenarios, robots in co-working space must understand the human intention for the robotic systems to make decisions that are both productive and safe. AR as an interface in human-robot co-working space is being actively researched to optimise safety, and productivity [126, 127, 128, 129, 130, 131].

We are looking at predicting the intent of human gestures from an egocentric point of view provided by AR devices. Understanding and predicting human intent in a co-working environment can lead to efficient communication between humans and robots to provide safety for humans and better planning for robotic systems. Collecting data from the real-world human-robot environment is not an easy task. There is ongoing work to provide a simulation environment that comprises of a co-working environment between humans and robots in a warehouse. Preliminary experiments performed using StAG LSTM(explained in Chapter 6) to predict intent from ego-hand gestures showed promising results both on simulated and real data. This work is being done in collaboration with the NVidia ISAAC team.

7.2.2 Possible Extensions

The research presented so far in the thesis moved towards using ego-hand gesture recognition in a real-world scenario. However, the head-mounted AR/VR devices have much less compute capabilities compared to a desktop. The networks designed so far need large GPUs with considerable memory to hold the network and run massive parallel convolutional operations. One possible extension to the current work would be to decrease the memory footprint and look if redundant operations could be eliminated and define a smaller network that can run on mobile GPU platforms to recognise ego-hand gestures on AR/VR devices. Computation time for recognising a gesture is also as important as the size of the network. The response time to the gesture performed determines the experience a user would have in the AR/VR environment. This response time has to be almost negligible for the user to have a good experience. In this regard the computation time for gesture recognition is of paramount importance. In addition to smaller memory footprint on the device, another direction that has to be explored is the real-time detection of gestures.

Devices like the latest Oculus Quest can locate ego-hands in space. The network we designed that generates embeddings for gestures and ego-hands simultaneously can take advantage of this information and could be adapted to recognise ego-hand gestures with their location as an additional input. The current network designed for recognising ego-hand gestures on untrimmed video does not pay attention to ego-hands in particular. The SSAR network can be adapted to pay attention to the ego-hands' motion and can further improve on CFJI and JI metrics, and early recognition than the current state of the art. The state activation function defined in StAG LSTM in the current module is a simple, fully connected neural network. It would be interesting to use different variations of this state activation function to address more complex scenarios like the presence of non-ego hands, ego-hands interacting with real objects, and adapt the network to online ego-action detection.

Gestures are task specific, meaning a set of gestures used in one application can be different from those used in another application. StAG LSTM architecture can be extended to identify the task/application at hand, route the input to the right task/application recognition network. This can give the user experience designers the ability to tailor gestures to a given task, in-turn giving the end user a better experience.

The Green Screen dataset currently provides only trimmed videos. We have conceptually and empirically shown that it is possible to train ego-hand gesture recognition algorithms on trimmed data by using green screen augmentation procedure. The Green Screen dataset can be extended for untrimmed videos. Finally, for consumer adaption of ego-hand gestures as an interface, they must be designed considering ergonomics. Ego-hand gesture recognition algorithms in conjunction with user studies on various gestures can lead to the design of ergonomic gestures paving the way to natural interfaces in AR and VR sans a joystick.

We want to conclude the thesis with the hope that the research done on ego-hand gesture recognition in terms of a new dataset, novel deep neural network architectures, extending the LSTM framework, a new loss function and a new evaluation metric would eventually lead to large scale adaption of ego-hand gestures as natural interfaces to AR and VR devices.

Bibliography

[1] S. Mann, “Humanistic computing:” wearcomp” as a new framework and application for intelligent signal processing,” *Proceedings of the Special Issue on Intelligent Signal Processing, IEEE*, 1998.

[2] T. Kanade and M. Hebert, “First-person vision,” *Proceedings of the IEEE*, 2012.

[3] O. Aghazadeh, J. Sullivan, and S. Carlsson, “Novelty detection from an ego-centric perspective,” *Compter Vision and Pattern Recognition*, 2011.

[4] A. Fathi, A. Farhadi, and J. M. Rehg, “Understanding egocentric activities,” *International Conference on computer vision*, 2011.

[5] K. M. Kitani, T. Okabe, Y. Sato, and A. Sugimoto, “Fast unsupervised ego-action learning for first-person sports videos,” *Computer Vision and Pattern Recognition*, 2011.

[6] T. Starner, B. Schiele, and A. Pentland, “Visual contextual awareness in wearable computing,” *Digest of Papers. Second International Symposium on Wearable Computers*, 1998.

[7] B. Schiele, N. Oliver, T. Jebara, and A. Pentland, “An interactive computer vision system dypers: Dynamic personal enhanced reality system,” *International Conference on Computer Vision Systems*, 1999.

[8] T. Starner, J. Weaver, and A. Pentland, “Real-time american

sign language recognition using desk and wearable computer based video," *IEEE Transactions on pattern analysis and machine intelligence*, 1998.

[9] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "EgoGesture: A New Dataset and Benchmark for Egocentric Hand Gesture Recognition," *IEEE Transactions on Multimedia*, vol. 9210, no. c, pp. 1–1, 2018.

[10] G. Serra, M. Camurri, L. Baraldi, M. Benedetti, and R. Cucchiara, "Hand segmentation for gesture recognition in ego-vision," *Proceedings of the 3rd ACM international workshop on Interactive multimedia on mobile & portable devices - IMMPD '13*, 2013.

[11] Y. Jang, I. Jeon, T.-K. Kim, and W. Woo, "Metaphoric hand gestures for orientation-aware vr object manipulation with an egocentric viewpoint," *IEEE Transactions on Human-Machine Systems*, 2016.

[12] C. Cao, Y. Zhang, Y. Wu, H. Lu, and J. Cheng, "Egocentric Gesture Recognition Using Recurrent 3D Convolutional Neural Networks with Spatiotemporal Transformer Modules," *2017 IEEE International Conference on Computer Vision (ICCV)*, pp. 3783–3791, 2017.

[13] H. G. Colt, S. W. Crawford, and O. Galbraith III, "Virtual reality bronchoscopy simulation: a revolution in procedural training," *Chest*, 2001.

[14] A. Gomes de Sá and G. Zachmann, "Virtual reality as a tool for verification of assembly and maintenance processes," *Computers and Graphics*, 1999.

[15] J. Vince, "Virtual reality techniques in flight simulation," *Virtual Reality Systems*, 1993.

[16] T. P. Caudell, "Introduction to augmented and virtual reality," *Proc.SPIE*, 1995.

[17] H. K. Wu, S. W. Y. Lee, H. Y. Chang, and J. C. Liang, "Current status, opportunities and challenges of augmented reality in education," *Computers and Education*, 2013.

[18] S. Nicolau, L. Soler, D. Mutter, and J. Marescaux, "Augmented reality in laparoscopic surgical oncology," *Surgical Oncology*, 2011.

[19] Z. Yovcheva, D. Buhalis, and C. Gatzidis, "Engineering augmented tourism experiences.," *Information and Communication Technologies in Tourism 2013*, 2013.

[20] W. Broll, J. Ohlenburg, I. Lindt, I. Herbst, and A.-K. Braun, "Meeting technology challenges of pervasive augmented reality games," *5th ACM SIGCOMM workshop on Network and system support for games*, 2006.

[21] "Saagie." `https://www.saagie.com/blog/object-detection-part1`.

[22] D. G. Lowe, "Distinctive image features from scale invariant keypoints," *International Journal of Computer Vision*, 2004.

[23] N. Dalal, B. Triggs, N. Dalal, B. Triggs, O. Gradients, and D. Cordelia, "Histograms of Oriented Gradients for Human Detection To cite this version : Histograms of Oriented Gradients for Human Detection," *International Conference on Computer Vision & Pattern Recognition (CVPR)*, 2005.

[24] H. Bay, T. Tuytelaars, and L. Van Gool, "SURF: Speeded up robust features," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2006.

[25] K. Jarrett, K. Kavukcuoglu, M. Ranzato, and Y. LeCun, "What is the best multi-stage architecture for object recognition?," *International Conference on Computer Vision*, 2009.

[26] H. Lee, R. Grosse, R. Ranganath, and A. Y. Ng, "Convolutional deep belief networks for scalable unsupervised learning of hi-

erarchical representations," *International Conference on Machine Learning*, 2009.

[27] Y. LeCun, K. Kavukcuoglu, and C. Farabet, "Convolutional networks and applications in vision," *International symposium on circuits and systems*, 2010.

[28] S. C. Turaga, J. F. Murray, V. Jain, F. Roth, M. Helmstaedter, K. Briggman, W. Denk, and H. S. Seung, "Convolutional networks can learn to generate affinity graphs for image segmentation," *Neural computation*, 2010.

[29] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "ImageNet Classification with Deep Convolutional Neural Networks," *Advances In Neural Information Processing Systems*, 2012.

[30] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A Large-Scale Hierarchical Image Database," *International Conference on Computer Vision & Pattern Recognition (CVPR)*, 2009.

[31] Y. LeCun, F. J. Huang, and L. Bottou, "Learning methods for generic object recognition with invariance to pose and lighting," *Proceedings of the 2004 IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 2004. CVPR 2004.*, 2004.

[32] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[33] "Wikipedia." `https://en.wikipedia.org/wiki/Convolutional_neural_network#Pooling_layer`.

[34] "Z2 little." `https://xzz201920.medium.com/conv1d-conv2d-and-conv3d-8a59182c4d6`.

[35] V. Nair and G. E. Hinton, "Rectified linear units improve re-

stricted boltzmann machines," *International Conference on Machine Learning*, 2010.

[36] P. J. Werbos, "Backpropagation Through Time: What It Does and How to Do It," *Proceedings of the IEEE*, 1990.

[37] Y. Bengio and P. Frasconi, "Credit assignment through time: Alternatives to backpropagation," *Advances in neural information processing systems*, 1994.

[38] J. L. Elman, "Finding structure in time," *Cognitive science*, 1990.

[39] S. E. Fahlman, "The recurrent cascade-correlation learning algorithm," *Advances in neural information processing systems*, 1991.

[40] R. J. Williams, "Complexity of exact gradient computation algorithms for recurrent neural networks," *Technical Report Technical Report NU-CCS-89-27, Boston: Northeastern ...*, 1989.

[41] K. Lang and G. Hinton, "The development of the time-delay neural network architecture for speech recognition," *Technical Report CMU-CS-88-152*, 1988.

[42] A. Waibel, T. Hanazawa, G. Hinton, K. Shikano, and K. J. Lang, "Phoneme recognition using time-delay neural networks," *IEEE transactions on acoustics, speech, and signal processing*, 1989.

[43] H. T. Siegelmann and E. D. Sontag, "On the computational power of neural nets," *Journal of computer and system sciences*, 1995.

[44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[45] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*. Springer, 2014.

[46] S. Hochreiter and J. Urgen Schmidhuber, “Long Short-Term Memory,” *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[47] P. Y. Simard, D. Steinkraus, and J. C. Platt, “Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis,” *International Conference on Document Analysis and Recognition*, 2003.

[48] D. Ciregan, U. Meier, and J. Schmidhuber, “Multi-column deep neural networks for image classification,” *Conference on Computer Vision and Pattern Recognition*, 2012.

[49] D. McNeill, *Hand and Mind: What Gestures Reveal About Thought*. University of Chicago Press, 1992.

[50] S. Tamura and S. Kawasaki, “Recognition of sign language motion images,” *Pattern Recognition*, 1988.

[51] A. Torige and T. Kono, “Human-Interface by Recognition of Human Gesture with Image processing Recognition of Gesture to Specify Moving Direction,” *IEEE Proceedings. of International Workshop on Robot and Human Communication*, pp. 105–110, 1992.

[52] T. Darrell and A. Pentland, “Space-time gestures,” *Computer Vision and Pattern Recognition, 1993. Proceedings CVPR ’93., 1993 IEEE Computer Society Conference on*, 1993.

[53] J. Davis and M. Shah, “Visual gesture recognition,” *IEE Proceedings: Vision, Image and Signal Processing*, 1994.

[54] K. Rangarajan and M. Shah, “Establishing motion correspondence,” *CVGIP: Image Understanding*, 1991.

[55] T. Starner and a. Pentland, “Real-time American Sign Language recognition from video using hidden Markov models,” *Computer Vision, 1995. Proceedings., International Symposium on*, 1995.

[56] J. Yamato, J. Ohya, and K. Ishii, “Recognizing human action in

time-sequential images using hidden Markov model," *Computer Vision and Pattern Recognition*, pp. 379–385, 1992.

[57] B.-w. Min, H.-s. Yoon, J. Soh, T. Ejimau, and I. Engineering, "Hand Gesture Recogrution Using Hidden Markov Models," *IEEE International Conference on Systems, Man, and Cybernetics. Computational Cybernetics and Simulation*, pp. 4232–4235, 1997.

[58] J. Segen and S. Kumar, "Human-computer interaction using gesture recognition and 3D hand\ntracking," *Proceedings 1998 International Conference on Image Processing. ICIP98 (Cat. No.98CB36269)*, 1998.

[59] W. Du and H. Li, "Vision based gesture recognition system with single camera," *Signal Processing Proceedings, 2000. WCCC-ICSP 2000. 5th International Conference on*, 2000.

[60] B. D. Zarit, B. J. Super, and F. K. Quek, "Comparison of five color models in skin pixel classification," *Proceedings - International Workshop on Recognition, Analysis, and Tracking of Faces and Gestures in Real-Time Systems, RATFG-RTS 1999*, 1999.

[61] H. S. Yoon, J. Soh, Y. J. Bae, and H. Seung Yang, "Hand gesture recognition using combined features of location, angle and velocity," *Pattern Recognition*, 2001.

[62] M. J. Swain and D. H. Ballard, "Color indexing," *International Journal of Computer Vision*, 1991.

[63] A. Ramamoorthy, N. Vaswani, S. Chaudhury, and S. Banerjee, "Recognition of dynamic hand gestures," *Pattern Recognition*, 2003.

[64] J. MacCormick and A. Blake, "A probabilistic contour discriminant for object localisation," *Computer Vision, 1998. Sixth International Conference on*, 1998.

[65] H. I. Suk, B. K. Sin, and S. W. Lee, "Hand gesture recogni-

tion based on dynamic Bayesian network framework," *Pattern Recognition*, 2010.

[66] J. Nagi and F. Ducatelle, "Max-pooling convolutional neural networks for vision-based hand gesture recognition," *International Conference on Signal and Image Processing Applications (ICSIPA)*, 2011.

[67] P. Barros, S. Magg, C. Weber, and S. Wermter, "A Multichannel Convolutional Neural Network for Hand Posture Recognition," *proceedings of 24th International Conference on Artificial Neural Networks, Lecture Notes in Computer Science Volume 8681,*, 2014.

[68] J. Triesch and C. V. D. Malsburgl, "Robust Classification of Hand Postures against Complex Backgrounds," *International Conference on Automatic Face and Gesture Recognition*, pp. 170–175, 1996.

[69] S. Ji, M. Yang, and K. Yu, "3D Convolutional Neural Networks for Human Action Recognition.," *PAMI*, 2013.

[70] A. Tang, K. Lu, Y. Wang, J. Huang, and H. Li, "A Real-Time Hand Posture Recognition System Using Deep Neural Networks," *ACM Trans. Intell. Syst. Technol.*, 2013.

[71] S. Escalera, X. Baró, J. Gonzàlez, M. Bautista, M. Madadi, M. Reyes, V. Ponce-López, H. Escalante, J. Shotton, and I. Guyon, """ChaLearn Looking at People Challenge 2014"," *ECCV Workshops*, 2014.

[72] N. Neverova, C. Wolf, G. W. Taylor, and F. Nebout, "Multi-scale Deep Learning for Gesture Detection and Localization," *ECCV Workshop*, 2014.

[73] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz, "Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 4207–4215, 2016.

[74] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," *Conference on Computer Vision and Pattern Recognition*, pp. 2625–2634, 2015.

[75] M. S. Ibrahim, S. Muralidharan, Z. Deng, A. Vahdat, and G. Mori, "Hierarchical Deep Temporal Models for Group Activity Recognition," *Conference on Computer Vision and Pattern Recognition*, pp. 1971–1980, 2016.

[76] X. Chen and C. L. Zitnick, "Learning a Recurrent Visual Representation for Image Caption," *Conference on Computer Vision and Pattern Recognition*, pp. 2422–2431, 2015.

[77] L. Liu and L. Shao, "Learning discriminative representations from RGB-D video data," *IJCAI International Joint Conference on Artificial Intelligence*, pp. 1493–1500, 2013.

[78] D. Lee, H. Yoon, and J. Kim, "Continuous gesture recognition by using gesture spotting," *International Conference on Control, Automation and Systems*, 2016.

[79] G. Zhu, L. Zhang, P. Shen, J. Song, S. A. A. Shah, and M. Bennamoun, "Continuous gesture segmentation and recognition using 3dcnn and convolutional lstm," *IEEE Transactions on Multimedia*, 2019.

[80] G. Benitez-Garcia, M. Haris, Y. Tsuda, and N. Ukita, "Continuous Finger Gesture Spotting and Recognition Based on Similarities Between Start and End Frames," *IEEE Transactions on Intelligent Transportation Systems*, 2020.

[81] A. F. Bobick and J. W. Davis, "The recognition of human movement using temporal templates," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2001.

[82] V. Buchmann, S. Violich, M. Billinghurst, and A. Cockburn, "FingARtips," *International Conference on Computer Graphics and Interactive Techniques*, 2004.

[83] ARTOOLKIT, "http://www.hitl.washington.edu/artoolkit/," *Community*, 2003.

[84] C. Li and K. M. Kitani, "Pixel-level hand detection in ego-centric videos," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2013.

[85] L. v. d. M. Maaten and G. Hinton, "Visualizing data using t-sne laurens," *Journal of Machine Learning Research*, 2008.

[86] R. Wen, W. L. Tay, B. P. Nguyen, C. B. Chng, and C. K. Chui, "Hand gesture guided robot-assisted surgery based on a direct augmented reality interface," *Computer Methods and Programs in Biomedicine*, 2014.

[87] C. Zimmermann and T. Brox, "Learning to Estimate 3D Hand Pose from Single RGB Images," *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[88] V. Jain, R. Perla, and R. Hebbalaguppe, "AirGestAR: Leveraging Deep Learning for Complex Hand Gestural Interaction with Frugal AR Devices," *Adjunct Proceedings of the 2017 IEEE International Symposium on Mixed and Augmented Reality, ISMAR-Adjunct 2017*, pp. 235–239, 2017.

[89] T. Malisiewicz, A. Gupta, and A. A. Efros, "Ensemble of exemplar-SVMs for object detection and beyond," *Proceedings of the IEEE International Conference on Computer Vision*, pp. 89–96, 2011.

[90] L. Baraldi, F. Paci, G. Serra, L. Benini, and R. Cucchiara, "Gesture recognition in ego-centric videos using dense trajectories and hand segmentation," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, pp. 702–707, 2014.

[91] S. Hegde, R. Perla, R. Hebbalaguppe, and E. Hassan, "GestAR : Real Time Gesture Interaction for AR with Egocentric View GestAR : Real Time Gesture Interaction for AR with Egocen-

tric View," *IEEE International Symposium on Mixed and Augmented Reality Adjunct Proceedings*, no. August, 2016.

[92] M. Abavisani, H. R. V. Joze, and V. M. Patel, "Improving the performance of unimodal dynamic hand-gesture recognition with multimodal training," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2019.

[93] J. Carreira and A. Zisserman, "Quo Vadis, action recognition? A new model and the kinetics dataset," *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017.

[94] Y. Zhang, L. Shi, Y. Wu, K. Cheng, J. Cheng, and H. Lu, "Gesture recognition based on deep deformable 3D convolutional neural networks," *Pattern Recognition*, 2020.

[95] S. Buch, V. Escorcia, B. Ghanem, L. Fei-Fei, and J. C. Niebles, "End-to-end, single-stream temporal action detection in untrimmed videos," *British Machine Vision Conference 2017, BMVC 2017*, 2017.

[96] X.-Y. Zhang, H. Shi, C. Li, K. Zheng, X. Zhu, and L. Duan, "Learning Transferable Self-Attentive Representations for Action Recognition in Untrimmed Videos with Weak Supervision," *Proceedings of the AAAI Conference on Artificial Intelligence*, 2019.

[97] O. Köpüklü, A. Gunduz, N. Kose, and G. Rigoll, "Real-time hand gesture detection and classification using convolutional neural networks," *Proceedings - 14th IEEE International Conference on Automatic Face and Gesture Recognition, FG 2019*, 2019.

[98] M. Asadi-Aghbolaghi, A. Clapes, M. Bellantonio, H. J. Escalante, V. Ponce-Lopez, X. Baro, I. Guyon, S. Kasaei, and S. Escalera, "A Survey on Deep Learning Based Approaches for Action and Gesture Recognition in Image Sequences," *IEEE International*

Conference on Automatic Face & Gesture Recognition (FG), 2017.

[99] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-Term Recurrent Convolutional Networks for Visual Recognition and Description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.

[100] L. Pigou, A. van den Oord, S. Dieleman, M. Van Herreweghe, and J. Dambre, "Beyond Temporal Pooling: Recurrence and Temporal Convolutions for Gesture Recognition in Video," *International Journal of Computer Vision*, 2018.

[101] B. Mahasseni and S. Todorovic, "Regularizing Long Short Term Memory with 3D Human-Skeleton Sequences for Action Recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016.

[102] M. Xu, M. Gao, Y. T. Chen, L. Davis, and D. Crandall, "Temporal recurrent networks for online action detection," *Proceedings of the IEEE International Conference on Computer Vision*, 2019.

[103] C. Li, P. Wang, S. Wang, Y. Hou, and W. Li, "Skeleton-based action recognition using LSTM and CNN," *2017 IEEE International Conference on Multimedia and Expo Workshops, ICMEW 2017*, 2017.

[104] R. Goyal, S. E. Kahou, V. Michalski, J. Materzynska, S. Westphal, H. Kim, V. Haenel, I. Fruend, P. Yianilos, M. Mueller-Freitag, F. Hoppe, C. Thurau, I. Bax, and R. Memisevic, "The 'Something Something' Video Database for Learning and Evaluating Visual Common Sense," *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[105] D. Damen, H. Doughty, G. M. Farinella, S. Fidler, A. Furnari, E. Kazakos, D. Moltisanti, J. Munro, T. Perrett, W. Price, and M. Wray, "Scaling Egocentric Vision: The EPIC-KITCHENS

Dataset," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2018.

[106] Y. Li, M. Liu, and J. M. Rehg, "In the Eye of Beholder : Joint Learning of Gaze and Actions in First Person Video," *Proceedings of the IEEE European Conference on Computer Vision*, 2018.

[107] J. Wan, S. Z. Li, Y. Zhao, S. Zhou, I. Guyon, and S. Escalera, "ChaLearn Looking at People RGB-D Isolated and Continuous Datasets for Gesture Recognition," *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2016.

[108] S. Bambach, S. Lee, D. J. Crandall, and C. Yu, "Lending a hand: Detecting hands and recognizing activities in complex egocentric interactions," *The IEEE International Conference on Computer Vision (ICCV)*, 2015.

[109] X. Zhu, W. Liu, X. Jia, and K.-Y. K. Wong, "A two-stage detector for hand detection in ego-centric videos," *2016 IEEE Winter Conference on Applications of Computer Vision (WACV)*, 2016.

[110] A. Urooj and A. Borji, "Analysis of hand segmentation in the wild," *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018.

[111] N. Navab, J. Hornegger, W. M. Wells, and A. F. Frangi, "U-net: Convolutional networks for biomedical image segmentation," *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015.

[112] S. Wu, S. Zhong, and Y. Liu, "Deep Residual Learning for Image Recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 1–17, 2016.

[113] Z. Wu, X. Wang, Y.-G. Jiang, H. Ye, and X. Xue, "Modeling spatial-temporal clues in a hybrid deep learning framework

for video classification," *Proceedings of the 23rd ACM International Conference on Multimedia*, 2015.

[114] J. Liu, G. Wang, P. Hu, L.-Y. Duan, and A. C. Kot, "Global context-aware attention lstm networks for 3d action recognition," *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017.

[115] W. Zhu, C. Lan, J. Xing, W. Zeng, Y. Li, L. Shen, and X. Xie, "Co-occurrence feature learning for skeleton based action recognition using regularized deep lstm networks," *Thirtieth AAAI Conference on Artificial Intelligence*, 2016.

[116] T.-Y. Lin, C. L. Zitnick, and P. Doll, "Microsoft COCO : Common Objects in Context," *Arixiv*, no. 1405.0312v3, pp. 1–15, 2015.

[117] A. M. Saxe, J. L. McClelland, and S. Ganguli, "Exact solutions to the nonlinear dynamics of learning in deep linear neural networks," *arXiv preprint arXiv:1312.6120*, 2013.

[118] X. Glorot and Y. Bengio, "Understanding the difficulty of training deep feedforward neural networks," *International conference on artificial intelligence and statistics*, pp. 249–256, 2010.

[119] R. R. Selvaraju, A. Das, R. Vedantam, M. Cogswell, D. Parikh, and D. Batra, "Grad-cam: Why did you say that? visual explanations from deep networks via gradient-based localization," *CoRR*, 2016.

[120] T. Chalasani and A. Smolic, "Simultaneous segmentation and recognition: Towards more accurate ego gesture recognition," *Proceedings of the IEEE International Conference on Computer Vision Workshops*, 2019.

[121] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3D convolutional networks," *Proceedings of the IEEE International Conference on Computer Vision*, 2015.

[122] Z. Qiu, T. Yao, and T. Mei, "Learning Spatio-Temporal Represen-

tation with Pseudo-3D Residual Networks," *Proceedings of the IEEE International Conference on Computer Vision*, 2017.

[123] D. Tran, H. Wang, L. Torresani, J. Ray, Y. Lecun, and M. Paluri, "A Closer Look at Spatiotemporal Convolutions for Action Recognition," *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2018.

[124] S. Wu, S. Zhong, and Y. Liu, "Deep Residual Learning for Image Recognition," *Computer Vision and Pattern Recognition*, 2016.

[125] J. Liu, A. Shahroudy, D. Xu, and G. Wang, "Spatio-temporal lstm with trust gates for 3d human action recognition," *European conference on computer vision*, 2016.

[126] L. Qian, A. Deguet, and P. Kazanzides, "ARssist: Augmented reality on a head-mounted display for the first assistant in robotic surgery," *Healthcare Technology Letters*, 2018.

[127] J. Guhl, J. Hügle, and J. Krüger, "Enabling human-robot-interaction via virtual and augmented reality in distributed control systems," in *Procedia CIRP*, 2018.

[128] F. De Pace, F. Manuri, A. Sanna, and C. Fornaro, "A systematic review of Augmented Reality interfaces for collaborative industrial robots," in *Computers and Industrial Engineering*, 2020.

[129] S. Papanastasiou, N. Kousi, P. Karagiannis, C. Gkournelos, A. Papavasileiou, K. Dimoulas, K. Baris, S. Koukas, G. Michalos, and S. Makris, "Towards seamless human robot collaboration: integrating multimodal interaction," *International Journal of Advanced Manufacturing Technology*, 2019.

[130] G. Avalle, F. De Pace, C. Fornaro, F. Manuri, and A. Sanna, "An Augmented Reality System to Support Fault Visualization in Industrial Robotic Tasks," *IEEE Access*, 2019.

[131] F. Muhammad, A. Hassan, A. Cleaver, and J. Sinapov, "Creating

a Shared Reality with Robots," *ACM/IEEE International Conference on Human-Robot Interaction*, 2019.