

SIGN LANGUAGE RECOGNIZER FRAMEWORK BASED ON DEEP LEARNING  
ALGORITHMS

By

Atra Akandeh

A DISSERTATION

Submitted to  
Michigan State University  
in partial fulfillment of the requirements  
for the degree of

Computer Science – Doctor of Philosophy

2021

## **ABSTRACT**

### **SIGN LANGUAGE RECOGNIZER FRAMEWORK BASED ON DEEP LEARNING ALGORITHMS**

By

Atra Akandeh

According to the World Health Organization (WHO, 2017), 5% of the world's population have hearing loss. Most people with hearing disabilities communicate via sign language, which hearing people find extremely difficult to understand. To facilitate communication of deaf and hard of hearing people, developing an efficient communication system is a necessity. There are many challenges associated with the Sign Language Recognition (SLR) task, namely, lighting conditions, complex background, signee body postures, camera position, occlusion, complexity and large variations in hand posture, no word alignment, coarticulation, etc.

Sign Language Recognition has been an active domain of research since the early 90s. However, due to computational resources and sensing technology constraints, limited advancement has been achieved over the years. Existing sign language translation systems mostly can translate a single sign at a time, which makes them less effective in daily-life interaction. This work develops a novel sign language recognition framework using deep neural networks, which directly maps videos of sign language sentences to sequences of gloss labels by emphasizing critical characteristics of the signs and injecting domain-specific expert knowledge into the system. The proposed model also allows for combining data from variant sources and hence combating limited data resources in the SLR field.

Copyright by  
ATRA AKANDEH  
2021

## **ACKNOWLEDGEMENTS**

It is a pleasure to thank those who made this thesis possible. I would like to thank my advisor Dr. Salem for his enlightening guidance and continued support throughout my entire Ph.D. program. In addition, I would like to thank my committee members Dr. Zhou, Dr. Aktulga, and Dr. Liu for their constructive guidance and valuable feedback. I would also like to express my gratitude to Dr. Torng, Dr. Esfahanian, and Dr. Kulkarni for their support. Finally, special thanks to my family and friends for their unconditional support and constant encouragement.



# TABLE OF CONTENTS

|   |      |
|---|------|
| LIST OF TABLES . . . . .                            | viii |
| LIST OF FIGURES . . . . .                           | ix   |
| CHAPTER 1 INTRODUCTION . . . . .                    | 1    |
| 1.1 Objective . . . . .                             | 2    |
| 1.2 Contribution . . . . .                          | 2    |
| 1.3 Thesis Overview . . . . .                       | 2    |
| CHAPTER 2 PRELIMINARIES . . . . .                   | 4    |
| 2.1 Deep Learning . . . . .                         | 4    |
| 2.2 CNNs . . . . .                                  | 6    |
| 2.3 RNNs . . . . .                                  | 7    |
| 2.4 Transfer Learning . . . . .                     | 9    |
| 2.5 Connectionist Temporal Classification . . . . . | 9    |
| 2.6 Video Classification . . . . .                  | 10   |
| 2.6.1 Related Work . . . . .                        | 11   |
| CHAPTER 3 SLIM LSTMS . . . . .                      | 19   |
| 3.1 Introduction . . . . .                          | 19   |
| 3.2 New Variants of the LSTM model . . . . .        | 20   |
| 3.3 Dataset . . . . .                               | 25   |
| 3.3.1 MNIST . . . . .                               | 25   |
| 3.3.1.1 The Tanh Activation . . . . .               | 26   |
| 3.3.1.2 The Sigmoid activation . . . . .            | 27   |
| 3.3.1.3 The ReLU activation . . . . .               | 28   |
| 3.3.2 IMDB . . . . .                                | 30   |
| 3.3.2.1 The Sigmoid activation . . . . .            | 30   |
| 3.3.3 20 Newsgroups . . . . .                       | 32   |
| 3.3.3.1 The Tanh activation . . . . .               | 34   |
| 3.4 Conclusion . . . . .                            | 45   |
| CHAPTER 4 FEATURE EXTRACTOIN MODEL . . . . .        | 46   |
| 4.1 MediaPipe Hands . . . . .                       | 46   |
| 4.2 MediaPipe Pose . . . . .                        | 47   |
| CHAPTER 5 CHARACTER-LEVEL SLR . . . . .             | 49   |
| 5.1 Introduction . . . . .                          | 49   |
| 5.1.1 Characteristics . . . . .                     | 50   |
| 5.1.2 Challenges . . . . .                          | 50   |
| 5.1.3 Related Work . . . . .                        | 51   |
| 5.2 Proposed Models . . . . .                       | 52   |

|  |                                |    |
|--|--------------------------------|----|
| 5.2.1                                  | Model Overview . . . . .       | 54 |
| 5.2.2                                  | Model Details . . . . .        | 55 |
| 5.3                                    | Dataset . . . . .              | 56 |
| 5.4                                    | Experimental Setup . . . . .   | 57 |
| 5.5                                    | Experimental Results . . . . . | 57 |
| 5.6                                    | Discussion . . . . .           | 59 |
| 5.7                                    | Summary . . . . .              | 59 |
| CHAPTER 6 WORD-LEVEL SLR . . . . .     |                                | 61 |
| 6.1                                    | Introduction . . . . .         | 61 |
| 6.1.1                                  | Characteristics . . . . .      | 62 |
| 6.1.2                                  | Challenges . . . . .           | 62 |
| 6.1.3                                  | Related Work . . . . .         | 62 |
| 6.2                                    | Proposed Models . . . . .      | 64 |
| 6.2.1                                  | Model Overview . . . . .       | 65 |
| 6.2.2                                  | Model Details . . . . .        | 66 |
| 6.2.2.1                                | RSign . . . . .                | 66 |
| 6.2.2.2                                | MCSign . . . . .               | 67 |
| 6.2.3                                  | Design Choice . . . . .        | 70 |
| 6.3                                    | Dataset . . . . .              | 71 |
| 6.4                                    | Experimental Setup . . . . .   | 72 |
| 6.5                                    | Experimental Results . . . . . | 72 |
| 6.6                                    | Discussion . . . . .           | 74 |
| 6.7                                    | Summary . . . . .              | 74 |
| CHAPTER 7 SENTENCE-LEVEL SLR . . . . . |                                | 76 |
| 7.1                                    | Introduction . . . . .         | 76 |
| 7.1.1                                  | Characteristics . . . . .      | 77 |
| 7.1.2                                  | Challenges . . . . .           | 77 |
| 7.1.3                                  | Related Work . . . . .         | 78 |
| 7.2                                    | Proposed Models . . . . .      | 79 |
| 7.2.1                                  | Model Overview . . . . .       | 80 |
| 7.2.2                                  | Model Details . . . . .        | 81 |
| 7.2.2.1                                | RSign-C . . . . .              | 81 |
| 7.2.2.2                                | MCSign-C . . . . .             | 82 |
| 7.2.3                                  | Design Choice . . . . .        | 84 |
| 7.3                                    | Dataset . . . . .              | 85 |
| 7.4                                    | Experimental Setup . . . . .   | 86 |
| 7.5                                    | Experimental Results . . . . . | 86 |
| 7.5.1                                  | MCSign-C & RSign-C . . . . .   | 86 |
| 7.5.2                                  | MCSign-C-Slim . . . . .        | 87 |
| 7.6                                    | Discussion . . . . .           | 88 |
| 7.7                                    | Summary . . . . .              | 89 |
| CHAPTER 8 FUTURE ROADMAP . . . . .     |                                | 90 |

|                        |    |
|------------------------|----|
| BIBLIOGRAPHY . . . . . | 91 |
|------------------------|----|

## LIST OF TABLES

|  |    |
|--|----|
| Table 3.1: MNIST - Network specifications . . . . .                                    | 26 |
| Table 3.2: MNIST - Best results obtained using <i>tanh</i> . . . . .                   | 30 |
| Table 3.3: MNIST - Best results obtained using <i>sigmoid</i> . . . . .                | 36 |
| Table 3.4: MNIST - Best results obtained using <i>relu</i> . . . . .                   | 39 |
| Table 3.5: IMDB - Network specifications. . . . .                                      | 39 |
| Table 3.6: IMDB - Best results obtained sigmoid. . . . .                               | 41 |
| Table 3.7: News20 - Network specifications. . . . .                                    | 42 |
| Table 3.8: News20 - Best results obtained using <i>tanh</i> . . . . .                  | 44 |
| Table 5.1: Network used to train Sign Language MNIST and ASL Fingerspelling A Skeleton | 57 |
| Table 6.1: RSign Network specifications. . . . .                                       | 67 |
| Table 7.1: RWTH-PHOENIX Setup Statistics . . . . .                                     | 85 |
| Table 7.2: Comparison between various approaches on the RWTH-PHOENIX dataset . . .     | 87 |
| Table 7.3: Time per epoch for each MCSign-C-Slim model. . . . .                        | 88 |

## LIST OF FIGURES

|  |    |
|--|----|
| Figure 2.1: A typical CNN architecture . . . . .   | 6  |
| Figure 2.2: AlexNet architecture Krizhevsky et al. (2012) . . . . .                            | 7  |
| Figure 2.3: HMDB51 Kuehne et al. (2011) . . . . .  | 11 |
| Figure 2.4: The video classifier architectures proposed by Karpathy et al. (2014) . . . . .    | 12 |
| Figure 2.5: Multiresolution CNN architecture Karpathy et al. (2014) . . . . .                  | 12 |
| Figure 2.6: Two-stream architecture for video classification Simonyan & Zisserman (2014) .     | 13 |
| Figure 2.7: The video classifier architecture proposed by Ng et al. (2015) . . . . .           | 14 |
| Figure 2.8: Long-term Recurrent Convolutional Network Donahue et al. (2017) . . . . .          | 15 |
| Figure 2.9: Comparing 2D & 3D convolution kernel Tran et al. (2015) . . . . .                  | 16 |
| Figure 2.10: C3D proposed by Tran et al. (2015) . . . . .                                      | 16 |
| Figure 2.11: TSN proposed by Wang et al. (2014) . . . . .                                      | 17 |
| Figure 2.12: The video classifier architecture proposed by Zhu et al. (2017) . . . . .         | 18 |
| Figure 3.1: MNIST - Training & Test accuracy, $\sigma = \tanh, \eta = 1e-4$ . . . . .          | 27 |
| Figure 3.2: MNIST - Training & Test accuracy, $\sigma = \tanh, \eta = 1e-4$ . . . . .          | 28 |
| Figure 3.3: MNIST - Training & Test accuracy, $\sigma = \tanh, \eta = 1e-3$ . . . . .          | 29 |
| Figure 3.4: MNIST - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1e-4$ . . . . . | 31 |
| Figure 3.5: MNIST - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 2e-3$ . . . . . | 32 |
| Figure 3.6: MNIST - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1e-4$ . . . . . | 33 |
| Figure 3.7: MNIST - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 2e-3$ . . . . . | 34 |
| Figure 3.8: MNIST - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1e-3$ . . . . . | 35 |
| Figure 3.9: MNIST - Training & Test accuracy, $\sigma = \text{relu}, \eta = 1e-4$ . . . . .    | 35 |

|   |    |
|---|----|
| Figure 3.10: MNIST - Training & Test accuracy, $\sigma = \text{relu}, \eta = 2e-3$ . . . . .                          | 37 |
| Figure 3.11: MNIST - Training & Test accuracy, $\sigma = \text{relu}, \eta = 1e-4$ . . . . .                          | 37 |
| Figure 3.12: MNIST - Training & Test accuracy, $\sigma = \text{relu}, \eta = 1e-3$ . . . . .                          | 38 |
| Figure 3.13: MNIST - Training & Test accuracy of different $\eta$ , $\text{lstm10}, \text{relu}$ . . . . .            | 38 |
| Figure 3.14: IMDB - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1e-4$ . . . . .                        | 40 |
| Figure 3.15: IMDB - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1e-4$ . . . . .                        | 40 |
| Figure 3.16: IMDB - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1.25e-5$ . . . . .                     | 41 |
| Figure 3.17: IMDB - Training & Test accuracy, $\sigma = \text{sigmoid}, \eta = 1e-5$ . . . . .                        | 42 |
| Figure 3.18: News20 - Training & Test accuracy, $\sigma = \text{tanh}, \eta = 1e-3$ . . . . .                         | 43 |
| Figure 3.19: News20 - Training & Test accuracy, $\sigma = \text{tanh}, \eta = 1e-3$ . . . . .                         | 43 |
| Figure 3.20: News20 - Training & Test accuracy, $\sigma = \text{tanh}, \eta = 1e-3$ . . . . .                         | 44 |
| Figure 4.1: MediaPipe hands tracking module output example . . . . .  | 47 |
| Figure 4.2: MediaPipe pose module output example MediaPipe (2019) . . . . .   | 48 |
| Figure 4.3: The proportions of the human body according to Vitruvius MediaPipe (2019) . . . . .                       | 48 |
| Figure 5.1: American Sign Language Alphabet Sign Language Club (2012) . . . . .                                       | 50 |
| Figure 5.2: The character-level sign recognizer architecture proposed by Li et al. (2015) . . . . .                   | 52 |
| Figure 5.3: Feature extraction using PCANet Aly et al. (2016) . . . . .   | 53 |
| Figure 5.4: The character-level sign recognizer architecture proposed by Aly et al. (2016) . . . . .                  | 53 |
| Figure 5.5: Successive binary depth images character 'G' Rioux-Maldague & Giguere (2014) . . . . .                    | 54 |
| Figure 5.6: The character-level sign recognizer architecture proposed by Rioux-Maldague<br>& Giguere (2014) . . . . . | 54 |
| Figure 5.7: Unsegmented hand gesture samples Oyedotun & Khashman (2017) . . . . .                                     | 54 |
| Figure 5.8: Finger joints capture by Real-Sense Huang et al. (2015) . . . . .   | 55 |

|  |    |
|--|----|
| Figure 5.9: ASL alphabet skeleton created from 2D coordinates of the hands using MediaPipe                           | 56 |
| Figure 5.10: VGG-19 architecture Jaworek-Korjakowska et al. (2019)   | 56 |
| Figure 5.11: Training & Validation Accuracy for the SConv model on Sign Language MNIST dataset.                      | 58 |
| Figure 5.12: Training & Validation Accuracy of theSVGG (left) & SConv (right) models on Sign Language MNIST dataset. | 58 |
| Figure 5.13: Confusion Matrix of SConv model on ASL Fingerspelling A datasets.                                       | 59 |
| Figure 6.1: Color, depth and skeleton images Huang et al. (2015)   | 63 |
| Figure 6.2: The word-level sign recognizer architecture proposed by Huang et al. (2015)                              | 63 |
| Figure 6.3: The word-level sign recognizer architecture proposed by Liu et al. (2016)                                | 64 |
| Figure 6.4: The word-level sign recognizer architecture proposed by Kumar et al. (2017)                              | 65 |
| Figure 6.5: MCSign Model Overview  | 66 |
| Figure 6.6: RSign model overview   | 67 |
| Figure 6.7: The skeleton joints of ASL sign “Christmas”  | 68 |
| Figure 6.8: MCSign Model Architecture  | 68 |
| Figure 6.9: MCSign model summary   | 69 |
| Figure 6.10: Optical flow field sample   | 70 |
| Figure 6.11: Frames with emphasis on hands   | 71 |
| Figure 6.12: ASLLVD Neidle et al. (2012)   | 71 |
| Figure 6.13: 40 common hand shapes used in ASL   | 73 |
| Figure 6.14: Training & Validation Accuracy for the MCSign model on ASLLV dataset.                                   | 73 |
| Figure 6.15: Training & Validation Accuracy for the RSign model on ASLLV dataset.                                    | 74 |
| Figure 7.1: The sentence-level sign recognizer architecture proposed by Koller et al. (2016a)                        | 78 |
| Figure 7.2: The sentence-level sign recognizer architecture proposed by Koller et al. (2016b)                        | 79 |

|  |    |
|--|----|
| Figure 7.3: A 2D black and white image sequence created from PHOENIX dataset . . . . . | 81 |
| Figure 7.4: MCSign-C model overview . . . . .  | 81 |
| Figure 7.5: RSign-C model overview . . . . .   | 82 |
| Figure 7.6: MCSign-C architecture . . . . .  | 83 |
| Figure 7.7: MCSign-C model summary . . . . .   | 84 |
| Figure 7.8: MCSign-C-Slim models comparison . . . . .                                  | 88 |



# **CHAPTER 1**

## **INTRODUCTION**

Deaf and hard of hearing people use sign language to communicate. Sign languages employ the visual-manual modality to convey meaning, and they are challenging to be understood by hearing people. Although making use of written communication or seeking help from a sign language interpreter can ease the situation, each has its drawbacks in terms of availability or convenience. To break the barrier between the hearing and the deaf, developing an efficient communication system is a necessity. The development of sign language translation technology dates back to the early 90s Fang et al. (2017). However, due to computational resources and sensing technology constraints, limited advancement has been achieved over the years.

Early Sign Language Recognition (SLR) systems performed data acquisition using sensor-based devices such as data gloves and accelerometers. These devices provide position, orientation, velocity, and other specifics of the hands. However, due to the prohibitive costs of such approaches, vision-based devices have been introduced. Microsoft Kinect, Leap Motion Controller, and Google Tango provide RGB and depth-maps information, which can be used as an image-based input to a framework.

Sign language is far more than just a collection of well-specified gestures, and many factors need to be taken into account when performing the Sign Language Recognition task. In character-level SLR, dealing with lighting conditions, complex background, signer body postures, camera position, and subtle differences between different letters is challenging. In word-level SLR, occlusion, complexity, large variations in hand posture, and variation in input length caused by signing speed need to be considered. In sentence-level SLR, there is no word alignment, and temporal boundaries of a specific word are not clear. Moreover, signs are context-dependent, and coarticulation in which sign is affected by the preceding or following signs plays an important role.

## **1.1 Objective**

Language translation technology is still far from being practically useful. Developing successful sign language recognition systems requires expertise in a wide range of fields, including computer vision, natural language processing, linguistics, and deaf culture, which is often overlooked by computer scientists Bragg et al. (2019). This work aims to provide a thorough study on SLR in three character, word and sentence levels, their characteristics, challenges associated with each, and how they have been approached previously.

Existing sign language recognition systems can only recognize a single sign at a time rather than sentence-level recognition, making them less effective in daily-life interaction. This work also develops a novel sign language recognition framework using deep neural networks, which directly maps videos of sign language sentences to sequences of gloss labels.

## **1.2 Contribution**

The contribution of this thesis is threefold: first, it thoroughly studies sign language recognition in three levels of character, word, and sentence. In all three levels, two common input scenarios, i.e., raw data and extracted features, have been explored and compared.

Second, it proposes a novel framework for sentence-level recognition consisting of a feature extractor piped into a deep sequential network. The framework is capable of real-time translation of American Sign Language into text.

Third, it combats limited data resources in the ASL field by combining different datasets from variant sources. Since the proposed network is insensitive to dataset type, combining different datasets boosts the network’s performance tremendously.

## **1.3 Thesis Overview**

In chapter 2, we review two significant architectures, namely CNN and RNN. We also describe transfer learning and connectionist temporal classification. We then present an overview of video classification problems and provide a literature review on approaches taken by researchers in

this domain. In chapter 3, to investigate and model the temporal dynamics of sign languages, we experiment with LSTM and implement slim variations of it proposed by Salem (2018). We perform an excessive empirical evaluation on different datasets comparing these variants with standard LSTM and obtain promising results. We then adapt what we learned in the SLR task. In chapter 4, we introduce Mediapipe, an ML solution for live and streaming media that we leveraged to extract discriminative information. In chapters 5, 6, and 7 character-level SLR, word-level SLR, and sentence-level SLR are investigated, respectively. Starting with characteristics and challenges associated with each one, we propose a novel architecture that successfully performs corresponding tasks. Finally, In chapter 8, we conclude this dissertation by addressing essential gaps in the context of sign languages and describe the possible directions for future work.

## **CHAPTER 2**

### **PRELIMINARIES**

In this chapter, an overview of the key concepts that the following chapters rely on has been provided. We first review deep learning in general and two major architectures, namely CNN and RNN. We also describe transfer learning and connectionist temporal classification. We then present an overview of video classification problems and provide a literature review on approaches taken by researchers in this domain.

### **2.1 Deep Learning**

The performance of discriminative models such as a classifier or regressor not only depends on the classifier algorithm itself but also on the choice of data representation. Therefore, the first step to fulfill the given task is to come up with a meaningful representation of data and then feed them into a supervised predictor. Traditionally people tried to extract discriminative information from the data by trial and error. They performed feature engineering and came up with hand design features. This approach needs specific domain knowledge, and it is application-dependent. That is why representation-learning algorithms play an essential role today. A representation learning algorithm aims to perform a feature extraction algorithm that is task non-specific and does not require human prior knowledge. There are many different approaches in representation learning, such as probabilistic models, auto-encoders, manifold learning, and deep networks.

In many data science tasks, we are dealing with high-dimensional data. Raw data is usually noisy and includes redundancy. As it was mentioned earlier, one way to capture critical aspects of data and extract discriminative information is to explicitly pre-process the data and then pass the extracted feature to the discriminative algorithm. However, new findings have shown that this is not how the neocortex, which is in charge of cognitive abilities, deals with high dimensional data. The human brain receives a myriad of sensory data every second. Rather than explicitly pre-processing them, the brain sends out the data through a hierarchy to learn to represent observations and

overcome the curse of dimensionality Lee & Mumford (2003). This key finding has inspired many people and resulted in the emergence of deep learning networks. The focus of this field is to extract relevant information through different levels of a network. Instead of human-engineered features, the network itself comes up with a representation at each layer (level) to represent information.

However, deep networks were rarely used due to convergence problems until 2006. In 2006 Geoff Hinton initiated representation learning using deep networks Hinton et al. (2006). At the time, researchers were not able to train deep networks using common approaches like backpropagation. To train a deep model, Hinton used unsupervised feature extraction and obtained a new transformation of data. Specifically, he used the Restricted Boltzmann Machine (RBM) to initialize the network parameters to avoid poor local minima. He intended to learn new representations of data one level at a time. For example, for a given input, one can generate a new representation within the first layer. Then this new representation can be used as an input to the second layer, and the procedure is repeated. The parameter associated with each layer can then be used as an initial value. One may perform fine-tuning the parameter later on using backpropagation or any other training algorithm. This unsupervised pre-training learns a hierarchy of features one level at a time. Hinton's method has been then quickly followed up by Bengio et al. (2006), Ranzato et al. (2006), Lee et al. (2007), Bengio (2009) and many more later. It has been shown that layer-wise stacking of extracted features often yields better representations in terms of classification error Larochelle et al. (2009). However, today there is no need to use the Restricted Boltzmann Machine as a building block of a deep network.

There are many applications associated with representation learning. Namely, Speech Recognition and Signal Processing, Object Recognition, Natural Language Processing, Multitasking and Transfer Learning, Domain Adaptation, and so on Bengio et al. (2013). Although deep networks are promising, training a deep architecture is not challenge-free. The first challenge is to establish a clear objective function or target for training Bengio et al. (2013). The second important point is proper initialization. Layer-wise pre-training has been suggested by many authors to avoid poor local minima. The choice of nonlinearity is also another critical factor. It has been shown that

using a rectifying linear unit as nonlinearity speeds up convergence dramatically. The number of variables to be learned in deep networks is relatively large. Hence, deep networks may require more epochs to learn the parameter thoroughly. Efficient GPU training allows one to train longer Bengio et al. (2013). The choice of hyperparameter is also essential. For example, the learning rate can be adaptive instead of being fixed. Applying the dropout trick has also been suggested by Hinton et al. (2012) to end up with more robust results.

## 2.2 CNNs

CNN's LeCun et al. (1999), which are inspired by the visual cortex of a cat, are building blocks of deep networks. There are many advantages associated with CNNs over multilayer perceptron networks introduced by Rosenblatt (1958). First, the number of trainable parameters is much fewer. Secondly, it offers invariance to shifting, scaling, and other forms of distortion. Third, the formation of the input data is preserved.

ConvNets consist of three different types of layers: convolution, pooling, and fully connected layers. In the convolutional layers, a kernel (local receptive field) is passed over the input to create a feature map for the next layer. By restricting the neural weights of one layer to a local receptive field, relevant features can be extracted automatically. Then, a pooling layer is applied to the feature map, followed by a fully connected layer. In deep networks, convolutions and max-pooling layers are stacked on top of each other many times.

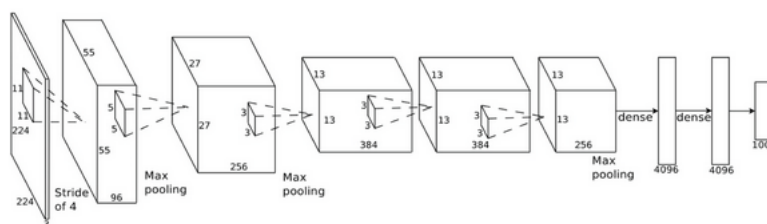


Figure 2.1: A typical CNN architecture

In 2012 Krizhevsky et al. (2012) introduced AlexNet, a new generation of convolutional networks which won ImageNet Large Scale Visual Recognition Challenge Deng et al. (2009). They

showed that the depth of the network is essential for high performance. They also exploited graphics processing units (GPUs) during training to circumvent computational cost. Later on similar architecture such as Inception Szegedy et al. (2014), VGG-Net Simonyan & Zisserman (2015), ResNet He et al. (2015), etc were introduced by others in the DL research community. The effectiveness of these networks makes convolutional networks to be the architecture of choice for image classification problems.

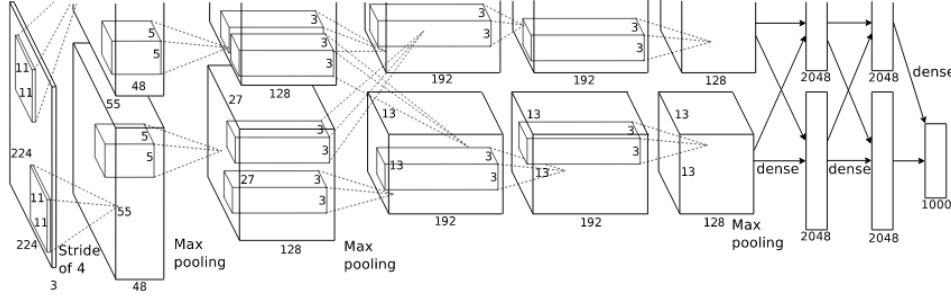


Figure 2.2: AlexNet architecture Krizhevsky et al. (2012)

## 2.3 RNNs

Recurrent Neural Networks (RNNs) have been making an impact in sequence-to-sequence mappings, with particularly successful applications in speech recognition, music, language translation, and natural language processing, to name a few Greff et al. (2017); Zaremba (2015); Chung et al. (2014); Boulanger-Lewandowski et al. (2012); Johnson et al. (2016). By their structure, they possess a memory (or state) and include feedback or recurrence. The *simple* RNN (sRNN) is succinctly expressed, see e.g., Goodfellow et al. (2016):

$$h_t = \sigma(W_{hx}x_t + W_{hh}h_{t-1} + b_h) \quad (2.1)$$

$$y_t = W_{hy}h_t + b_y$$

where  $x_t$  is the input sequence vector at (time) step  $t$ ,  $h_t$  is the hidden (activation) unit vector at step  $t$ , while  $h_{t-1}$  is the hidden unit vector at the previous step  $t - 1$ , and  $y_t$  is the output vector at step  $t$ . The parameters are the three matrices, namely,  $W_{hx}$ ,  $W_{hh}$ , and  $W_{hy}$ , and the vector  $b_h$ . This

constitutes a discrete-step dynamic recurrent system with  $h_t$  acting as the state. The parameters are to be determined adaptively via training mostly using various versions of backpropagation through time (BPTT), e.g., see Greff et al. (2017).

The LSTM RNNs introduce a cell-memory and three gating signals to enable effective learning via the BPTT Greff et al. (2017). The simple activation state has been replaced with a more involved activation with gating mechanisms. The LSTM RNN uses an additional memory cell (vector) and includes three gates: (i) an input gate,  $i_t$  (ii) an output gate  $o_t$ , and (iii) a forget gate,  $f_t$ . These gates collectively control signaling. The *standard* LSTM is expressed mathematically as Greff et al. (2017); Goodfellow et al. (2016):

$$\begin{aligned}
i_t &= \sigma_{in}(W_i x_t + U_i h_{t-1} + b_i) \\
f_t &= \sigma_{in}(W_f x_t + U_f h_{t-1} + b_f) \\
o_t &= \sigma_{in}(W_o x_t + U_o h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{2.2}$$

where the first four equations are a replica of the simple RNN (sRNN) above, with the first three equations serving as gating signals, and thus their nonlinear activation is set as a sigmoid function  $\sigma_{in}$ . The 4th equation's nonlinearity is arbitrary, typically sigmoid, hyperbolic tangent ( $\tanh$ ), or rectified linear unit (ReLU). This 4th equation is sometimes referred to as the input block. The last two equations entail the memory cell  $c_t$  and now the activation hidden unit  $h_t$  with the insertion of the gating signals in a point-wise (Hadamard) multiplications (using the symbol  $\odot$ ). This represents a discrete-step nonlinear dynamic system with recurrence. The distinct parameters are associated with each replica as  $W_*$ ,  $U_*$ , and  $b_*$  in a straight fashion.

The output layer of the LSTM model may be chosen to be as a linear (more accurately, affine) map as



$$y_t = W_{hy}h_t + b_y \quad (2.3)$$

where  $y_t$  is the output,  $W_{hy}$  is a matrix, and  $b_y$  is a bias vector. In other optional implementations, this layer may be followed by a softmax layer to render the output analogous with probability ranges.

## 2.4 Transfer Learning

Transfer-Learning is a deep learning technique where models trained successfully on larger datasets are reused on more specific data. This is done by freezing the weights at deeper layers from the pre-trained model and fine-tune weights at shallower layers. The weights can also be used as the starting point for the training process and adapted to the new problem. The main advantages of such a technique are its less demanding time and data requirements. Many high-performing models have been developed so far (e.g., VGG, GoogLeNet, Resnet). These models are pre-trained on imageNet Deng et al. (2009) and are widely used for transfer learning.

## 2.5 Connectionist Temporal Classification

Connectionist Temporal Classification (CTC), introduced by Graves et al. (2006), is a neural network layer and an associated scoring mechanism to tackle sequence to sequence problems when there is no alignment between input and output. In the CTC layer, a probability distribution over all labels at each time step is predicted. To calculate the probability of an output sequence, CTC works by summing over the probability of all possible alignments equivalent to a given label. Equivalent alignments of a given label are those in which collapsing repeating characters results in that label. For example, the label sequence 'ab' is defined as equivalent to the label sequences 'aaab', 'aabb', 'abbb'. To take account of repetition in the label sequence, CTC introduced the "blank" token. Blank tokens separate individual characters so that only repeated characters that are not separated by the blank are collapsed. For example, the label sequence 'aab' is defined as equivalent to the label sequences 'aa-ab', 'a-abb', 'a-aab', etc. CTC scores can then be used with the back-propagation algorithm to update the neural network weights.

The predicted probability distribution can also be used to infer a likely output. One intuitive solution could be finding the most likely output. However, this solution does not account for the fact that some alignments collapse to the same output. For example, the alignments 'aa-' and 'aaa' can individually have a lower probability than 'bbb', but the sum of their probabilities can be greater than that of 'bbb'. To address this issue, there are more advanced approaches such as beam-search decoding, prefix-search decoding, or token passing.

## **2.6 Video Classification**

After demonstrating outstanding results on image recognition problems, the next phase is finding an architecture to classify and analyze the semantic content of videos robustly. However, this is a harder task at hand. Videos have an additional temporal dimension, they are much larger in size, and they may contain different numbers of frames. An intuitive way to extend image-based CNN structures to the video domain is to perform classification on each frame independently and then conduct a later fusion, such as average scoring, to predict the action class of the video. In the next chapter, we thoroughly examine techniques and methods that different authors have employed to tackle these issues. UCF101 Soomro et al. (2012), Sports-1M Karpathy et al. (2014) and HMDB51 Kuehne et al. (2011) have been largely used by researchers to evaluate their models.

UCF101 is an action recognition dataset collected from YouTube. It contains 13,320 videos from 101 action categories, with super categories of Human-Object Interaction, Body-Motion, Human-Human Interaction, Playing Musical Instruments, and Sports. The Sports-1M is a large-scale dataset containing 1,133,158 video URLs that have been annotated automatically with 487 Sports labels. HMDB51 contains 6849 videos distributed in 51 action classes. It has been collected from various sources such as movies, Prelinger Archive, YouTube, and Google videos. Action categories include general facial actions, facial actions with object manipulation, general body movements, body movements with object interaction, and body movements for human interaction.

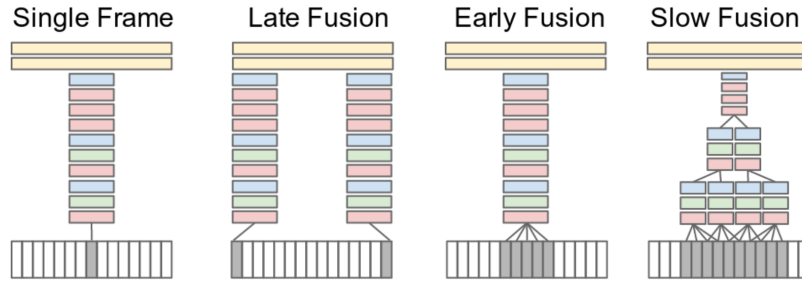


Figure 2.3: HMDB51 Kuehne et al. (2011)

### 2.6.1 Related Work

Karpathy et al. (2014) empirically studied various CNN Architecture on the Sport-1M and UCF-101 datasets. They proposed three different architectures to model temporal connectivity, namely late fusion, early fusion, and slow fusion networks (Figure 2.4). They first experimented with a single frame model as a baseline to perceive the contribution of static information to final predictions. Single frame architecture is similar to AlexNet model pre-trained with ImageNet. The late fusion model combined the information from two single frame networks (up to the last convolution layer) with shared parameters. Two pre-trained nets are placed 15 frames apart, and the merged output is then fed into two successive fully connected layers. Therefore, the first fully connected layer can compute global motion characteristics Karpathy et al. (2014). In the early fusion model, the architecture is similar to the single frame network, except that the first layer filters are modified to be of size  $11 \times 11 \times 3 \times T$ . Therefore, the information is combined across an entire time window (clip) early on. They set  $T$  to be 10 (there are ten frames in each clip) and trained the network from scratch. In the slow fusion model, first level filters of size  $11 \times 11 \times 3 \times 4$  with stride 2 generate four responses in time. The process is repeated in the second and third levels with filters of size  $11 \times 11 \times 3 \times 2$  and stride 2. Thus, the third convolutional layer has access to information across all ten input frames.

To speed up the training process, they adopted a multi-resolution architecture. A two-stream network of input size of  $89 \times 89 \times 3$ , namely, context stream for low-resolution frames and a fovea



consists of a two-stream network: one for spatial and the other for temporal content (figure 2.6). Both networks are mostly CNN-M-2048 introduced by Jia et al. (2014). The input to the spatial net is of size  $224, 224, 3$ , and the input to the temporal net is of size  $224, 224, 2L$ . In the temporal network, the consecutive grayscale frames are considered as a channel dimension in the convolution layer.

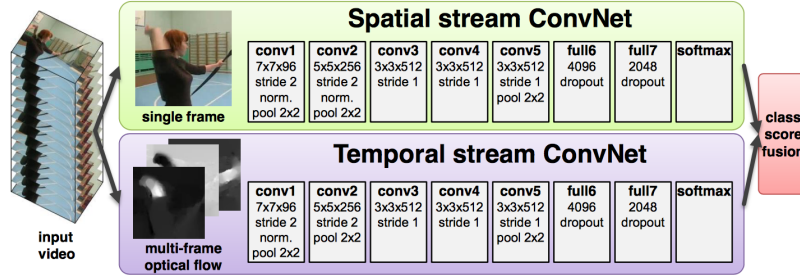


Figure 1: Two-stream architecture for video classification.

Figure 2.6: Two-stream architecture for video classification Simonyan & Zisserman (2014)

In the training phase, to deal with different time lengths of videos, a frame is randomly picked from each video. The frame is then fed into the spatial net. The input to the temporal net is a volume of size  $224 \times 224 \times 2L$ . They found out that setting  $L$  equal to 10 and placing horizontal and vertical components of flow alternatively in the stack delivers the best results. To combine temporal and spatial prediction scores, they tried averaging and also training a multi-class SVN. In the testing phase, they sampled 25 equal temporal spacing frames and propagated them through the two-stream networks. The final answer is then averaged across all samples.

Due to the insufficient size of the dataset, to avoid overfitting, one can combine two datasets to increase the amount of training data. However, intersections between two datasets may cause problems. To circumvent this issue, they performed multi-task learning by placing two softmax layers at the top. They were able to achieve an accuracy of 87% on UCF-101.

Similar to the previous paper, to obtain video-level prediction, they performed sampling and averaging. However, in this method, some samples may include partial or even no part of the actual action, and this may cause false labeling. Also, due to sampling, the task of long-range temporal information modeling was still unaccomplished.

Ng et al. (2015) explored the idea of extracting features using a CNN and then aggregating frame information in a separate network. They proposed two aggregation methods. In the first method, they experimented with various pooling operations and found out that max-pooling outperforms other pooling methods in terms of speed and accuracy. In the second method, they passed extracted features to a separate LSTM. They experimented with a various number of layers and memory cells and decided to use five layers architecture with 512 cell units. They also studied multiple strategies to combine LSTM frame-level predictions into a single video-level prediction.

They fine-tuned pre-trained GoogLeNet Szegedy et al. (2014) on optical flow and raw images and aggregated the results using the late fusion method. Given the nature of the method presented in their paper, prediction can be made with no need to sample. However, they found out sampling and averaging improve the results. In the training phase, they sampled 300 frames per video. Frames are repeated from the start for shorter videos. For data augmentation, multiple examples per video are obtained by randomly selecting the position of the first frame. They evaluated their proposed architectures on the UCF101 and Sport 1M datasets and were able to achieve 88.6% and 73.1%, respectively.

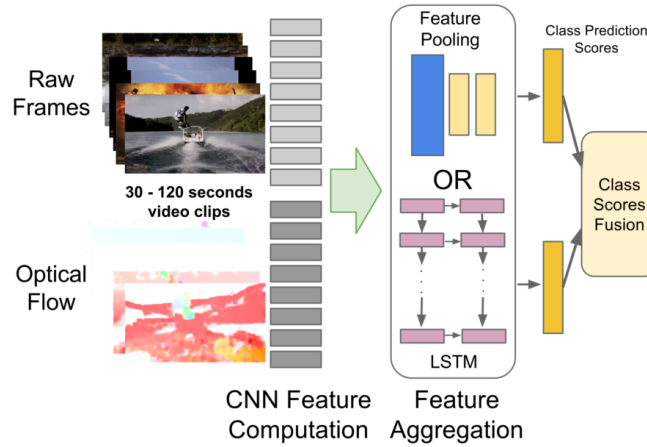


Figure 1: Overview of our approach.

Figure 2.7: The video classifier architecture proposed by Ng et al. (2015)

Donahue et al. (2017) explored three tasks in their paper: activity recognition, image captioning, and video description. In the first task, they combined temporal and spatial content into a single

network by taking advantage of an encoder-decoder idea. In the LRCN network, a CNN model as a feature extractor is applied to each frame before feeding them into an LSTM (figure 2.8). To combine temporal information, they took a late fusion approach; they averaged the output of each LSTM, which corresponds to each frame and then chose the most probable label. As can be seen, the convolution blocks serve as an encoder, and LSTM blocks serve as a decoder. The CNN Model they adopted was a combination of CaffeNet, AlexNet, and the network used by Zeiler & Fergus (2014). The network is pre-trained on ImageNet. The input to the network is RGB and optical flow field images. Final video-level prediction is then obtained by calculating the weighted average in favor of the flow network. They subsample each video to 16 frames, which prevents long-range temporal information modeling, as each video is generally on the order of 100 frames.

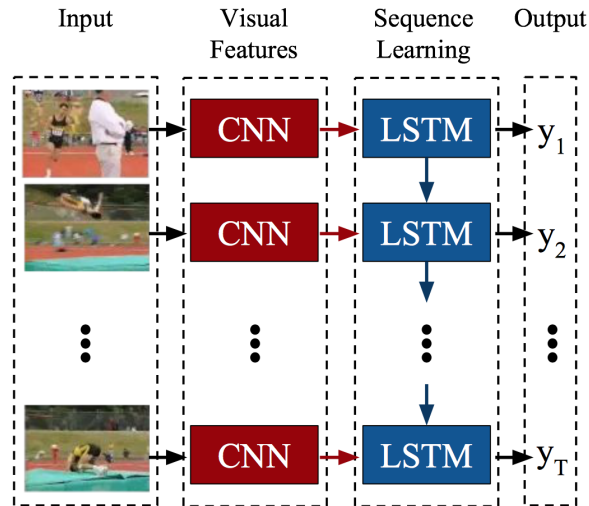


Figure 2.8: Long-term Recurrent Convolutional Network Donahue et al. (2017)

They evaluated their model on the UCF-101 dataset and compared it to a single frame baseline model. As mentioned before, in a single frame model, each frame is fed into a CNN and then averaged across all frames. They were able to obtain 82.92% accuracy. They reported that hidden unit size is the most effective factor in terms of accuracy, and that aggressive dropout is also beneficial.

Tran et al. (2015) explored the idea of using 3D convolution networks as a feature extractor

and spatiotemporal feature learner (figure 2.9). Their architecture consists of 8 convolution, five max-pooling, and two fully connected layers, followed by a softmax function (figure 2.10). They performed an extensive search to find a good architecture by concentrating on the first layer kernel size. They kept the spatial receptive field fixed at  $3 \times 3$  and experimented with the temporal depth of the 3D convolution kernels. They found out that  $3 \times 3 \times 3$  is the best kernel choice. To learn

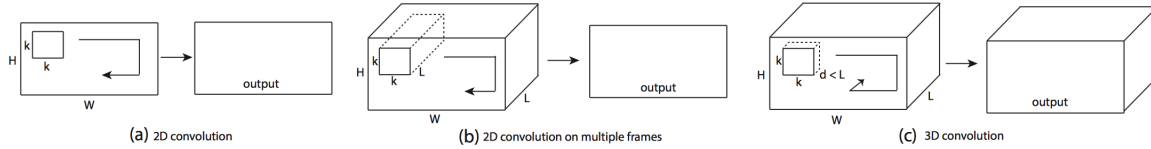


Figure 2.9: Comparing 2D & 3D convolution kernel Tran et al. (2015)

spatiotemporal features, they trained their model on the Sport-1M dataset. Each video is split into 16 non-overlapping clips in the training phase and used as an input to the network. Jittering and random crops are used as data augmentation. In the testing phase, ten random clips from each video are fed to the network, and the video-level prediction is then calculated by averaging across all clips. They trained their network from scratch and also fine-tuned it from the 1380K model.

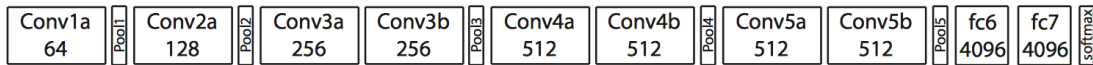


Figure 2.10: C3D proposed by Tran et al. (2015)

Next, they used C3D as a feature extractor on the UCF101 dataset and propagated the output to a multi-class linear SVN. They observed that the C3D extractor outperforms iDT features. Also, combining C3D features and iDT boosts the performance, as the high-level semantic information extractor C3D and low-level gradient extractor iDT are highly complementary to each other. They were able to obtain 90.4% and 85.2% accuracy with and without iDt features. They also used a deconvolution layer to decode what C3D is learning internally. They found out that the network initially concentrates on spatial information in the first few frames and the salient motion in the subsequent frames.



Wang et al. (2016), addressed the two-stream network, mentioned earlier, inability to capture long-range temporal information. They introduced a temporal segmentation network that receives evenly distributed snippets rather than random clips. They modified the two-stream network by splitting videos into segmentations. Each snippet is then chosen from each segment. Then Segmental Consensus is derived for each modality (RGB images, optical flow field, etc). They empirically evaluated Segmental Consensus functions (even averaging, maximum, weighted averaging) and decided to use even averaging for their architecture. Probability scores from each modality are then weight averaged to produce the video-level prediction (figure 2.11).

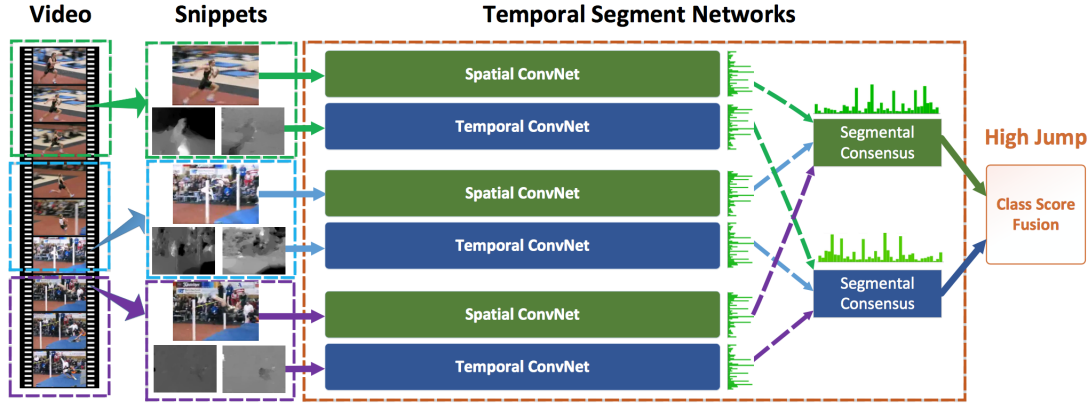


Figure 2.11: TSN proposed by Wang et al. (2014)

They set  $k$  (number of segmentations) equal to 3 according to previous works on temporal modeling by Gaidon et al. (2013) & Wang et al. (2014), and adopted the Inception model with Batch Normalization (BN-Inception), by Ioffe & Szegedy (2015), pre-trained on ImageNet as building blocks of two-stream networks. To benefit from the pre-trained network on optical flow, they scaled up the input to range 0 – 255 and averaged the weights across the RGB channels, and replicated it by the number of channels in the temporal network input. They studied the effect of adding additional modalities such as RGB difference and warped optical flow to extract salient motion and mitigate the effect of camera movement. They also benefited from data augmentation, heavy dropout, and batch normalization. They evaluated their model on the HMDB51 and UCF101 datasets and achieved 69.4% and 94.2%, respectively.

Zhu et al. (2017) introduced an end-to-end two-stream-based network. To speed up the process and avoid extra storage, the network generates an optical flow field rather than being fed by pre-computed optical flow images. Optical flow generation can be regarded as an image reconstruction problem. For two adjacent frames  $f_1$  and  $f_2$ , they generated flow  $V$  and then reconstructed  $f_2$  using  $f_1$  and  $V$  employing inverse warping by minimizing reconstruction error. They stacked this network, calling it MotionNet, on top of a temporal stream, treating it as an off-the-shelf flow estimator (figure 2.12). This modification resulted in more than 20x speedup compared to the traditional two-stream approaches.

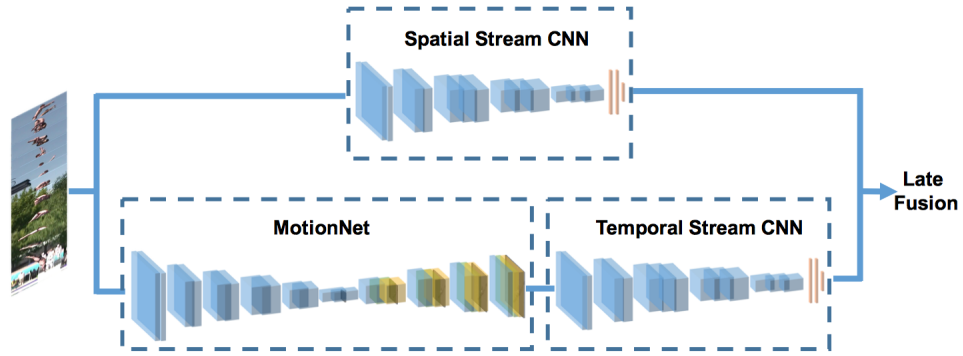


Figure 2.12: The video classifier architecture proposed by Zhu et al. (2017)

In the training phase, they evenly sampled 25 frames per video and performed 10x augmentation by flipping and cropping and then averaging the prediction scores (before softmax operation) over all crops of the samples. They evaluated their network on the UCF101 dataset and were able to obtain 89.2% accuracy. They improved their performance even further to 92.5% by fusing spatial and temporal streams using the TSN method.

## CHAPTER 3

### SLIM LSTMS

In this chapter, we implement and evaluate nine slim variants of the standard Long Short-term Memory (LSTM) recurrent neural networks proposed by Salem (2018). For simplicity, we refer to these models as LSTM1, LSTM2, LSTM3, LSTM4, LSTM5, LSTM4a, LSTM5a, LSTM6, LSTM10 & LSTM11. In the author’s most recent work, the first seven models have been referred to as LSTM-1 RNN, LSTM-2 RNN, LSTM-3 RNN, LSTM-4 RNN, LSTM-5 RNN, LSTM-4i RNN, LSTM-5i RNN, and LSTM-6 RNN. These variants have been generated by uniformly reducing blocks of adaptive parameters in the gating mechanisms. Such parameter-reduced variants enable speeding up data training computations and would be more suitable for implementations onto constrained embedded platforms. We comparatively evaluate and verify these variant models on the three classical datasets and demonstrate that these variant models are comparable to a standard implementation of the LSTM model while using fewer parameters. The implementation of the models is publicly accessible through Akandeh (2019).

### 3.1 Introduction

As mentioned earlier, the standard LSTM is expressed mathematically as

$$\begin{aligned} i_t &= \sigma_{in}(W_i x_t + U_i h_{t-1} + b_i) \\ f_t &= \sigma_{in}(W_f x_t + U_f h_{t-1} + b_f) \\ o_t &= \sigma_{in}(W_o x_t + U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \sigma(c_t) \end{aligned} \tag{3.1}$$

where  $\sigma_{in}$  is called inner activation (logistic) function which is bounded between 0 and 1, and  $\odot$  denotes point-wise multiplication. The output layer of the LSTM model may be chosen to be as a

linear map, namely,

$$y_t = W_{hy}h_t + b_y \quad (3.2)$$

LSTMs can be viewed as composed of the cell network and its three gating networks. LSTMs are relatively slow due to the fact that they have four sets of "weights," of which three are involved in the gating mechanism. In the following sections, we describe and demonstrate the comparative performance of nine simplified LSTM variants by removing select blocks of adaptive parameters from the gating mechanism and demonstrate that these variants are a competitive alternative to the original LSTM model while requiring a less computational cost.

### 3.2 New Variants of the LSTM model

LSTM uses a gating mechanism to control the signal flow. It possesses three gating signals driven by three main components, namely, the external input signal, the previous state, and a bias. We have proposed nine variants of the LSTM model, aiming at reducing the number of (adaptive) parameters in each gate and thus reduce computational cost Salem (7.11.2016). The first three models have been demonstrated previously in initial experiments in Lu & Salem (2017).

#### LSTM1

In this first model variant, input signals and their corresponding weights, namely, the terms  $W_ix_t, W_fx_t, W_ox_t$  have been removed from the equations in the three corresponding gating signals. The resulting result model becomes

$$\begin{aligned} i_t &= \sigma_{in}(U_i h_{t-1} + b_i) \\ f_t &= \sigma_{in}(U_f h_{t-1} + b_f) \\ o_t &= \sigma_{in}(U_o h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\ c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\ h_t &= o_t \odot \sigma(c_t) \end{aligned} \quad (3.3)$$

## LSTM2

In this second model variant, the gates have no bias and no input signals  $W_ix_t, W_fx_t, W_ox_t$ . Only the state is used in the gating signals. This produces

$$\begin{aligned}i_t &= \sigma_{in}(U_i h_{t-1}) \\f_t &= \sigma_{in}(U_f h_{t-1}) \\o_t &= \sigma_{in}(U_o h_{t-1}) \\ \tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\h_t &= o_t \odot \sigma(c_t)\end{aligned} \tag{3.4}$$

## LSTM3

In the third model variant, the only term in the gating signal is the (adaptive) bias. This model uses the least number of parameters among other variants.

$$\begin{aligned}i_t &= \sigma_{in}(b_i) \\f_t &= \sigma_{in}(b_f) \\o_t &= \sigma_{in}(b_o) \\ \tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\h_t &= o_t \odot \sigma(c_t)\end{aligned} \tag{3.5}$$

## LSTM4

In the fourth model variant, the  $U_i, U_f, U_o$  matrices have been replaced with the corresponding  $u_i, u_f, u_o$  vectors in LSTM2. The intent is to render the state signal with a point-wise multiplication.

Thus, one reduces parameters while retaining state feedback in the gatings.

$$\begin{aligned}
i_t &= \sigma_{in}(u_i \odot h_{t-1}) \\
f_t &= \sigma_{in}(u_f \odot h_{t-1}) \\
o_t &= \sigma_{in}(u_o \odot h_{t-1}) \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.6}$$

## LSTM5

In the fifth model variant, we revise LSTM1 so that the matrices  $U_i, U_f, U_o$  are replaced with corresponding vectors denoted by small letters. Then, as in LSTM4, we acquire (Hadamard) point-wise multiplication in the state variables.

$$\begin{aligned}
i_t &= \sigma_{in}(u_i \odot h_{t-1} + b_i) \\
f_t &= \sigma_{in}(u_f \odot h_{t-1} + b_f) \\
o_t &= \sigma_{in}(u_o \odot h_{t-1} + b_o) \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.7}$$

## LSTM4a

In LSTM4a, a fixed real number with an absolute value less than one has been set for the forget gate in order to preserve bounded-input-bounded-output (BIBO) stability, Salem (2016). Meanwhile,

the output gate is set to 1 (which in practice eliminates this gate altogether).

$$\begin{aligned}
i_t &= \sigma_{in}(u_i \odot h_{t-1}) \\
f_t &= c \\
o_t &= 1.0 \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.8}$$

### LSTM5a

LSTM5a is similar to LSTM4a, but the bias term in the input gate equation is preserved.

$$\begin{aligned}
i_t &= \sigma_{in}(u_i \odot h_{t-1} + b_i) \\
f_t &= c \\
o_t &= 1.0 \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.9}$$

### LSTM6

Finally, this is the most aggressive parameter reduction. Here all gating equations are replaced by an appropriate constant. For BIBO stability, we found that the forget gate must be set  $f_t = 0.59$  or below. The other two gates are set to 1 each (which practically eliminates them for computational efficiency purposes). In fact, this model variant now becomes equivalent to the so-called basic

RNN model reported in Salem (2016).

$$\begin{aligned}
i_t &= 1.0 \\
f_t &= \alpha \\
o_t &= 1.0 \\
\tilde{c}_t &= \sigma(W_c x_t + U_c h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.10}$$

### LSTM10

In this model, pointwise multiplication is applied to the hidden state and corresponding weights in the cell-body equations as well. We apply this modification not only to the gating equations but also to the main equation, i.e. matrix  $U_c$  is replaced with vector  $u_c$  for the pointwise multiplication.

$$\begin{aligned}
i_t &= \sigma_{in}(u_i \odot h_{t-1}) \\
f_t &= \sigma_{in}(u_f \odot h_{t-1}) \\
o_t &= \sigma_{in}(u_o \odot h_{t-1}) \\
\tilde{c}_t &= \sigma(W_c x_t + u_c \odot h_{t-1} + b_c) \\
c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
h_t &= o_t \odot \sigma(c_t)
\end{aligned} \tag{3.11}$$



## LSTM11

This variant is similar to the LSTM10. However, it reinstates the biases in the gating signals. Mathematically, it is expressed as

$$\begin{aligned}i_t &= \sigma_{in}(u_i \odot h_{t-1} + b_i) \\f_t &= \sigma_{in}(u_f \odot h_{t-1} + b_f) \\o_t &= \sigma_{in}(u_o \odot h_{t-1} + b_o) \\ \tilde{c}_t &= \sigma(W_c x_t + u_c \odot h_{t-1} + b_c) \\c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\h_t &= o_t \odot \sigma(c_t)\end{aligned}\tag{3.12}$$

## 3.3 Dataset

We train and evaluate our models on three benchmark datasets, namely MNIST, IMDB & 20 Newsgroups dataset. To avoid clutter in images, we plot performance in three sub-groups. First group contains LSTM, LSTM1, LSTM2, LSTM3, LSTM4 & LSTM5. Second groups depicts LSTM, LSTM4, LSTM5, LSTM4a & LSTM5a. Last group compares LSTM, LSTM6, LSTM10 & LSTM11.

### 3.3.1 MNIST

The MNIST dataset contains  $28 \times 28$  handwritten digits images. Treating images as row-wise sequences, each model reads one row at a time from top to bottom to produce its output after seeing all 28 rows. Table 3.1 provides a specification of the network used.

Three different nonlinearities, i.e., *tanh*, *sigmoid*, and *relu*, have been employed in the first (LSTM) layer. For each case, we perform parameter tuning over different values of the learning parameter.

Table 3.1: MNIST - Network specifications

|                        |                           |
|------------------------|---------------------------|
| Input dimension        | $28 \times 28$            |
| Number of hidden units | 100                       |
| Nonlinear function     | tanh, sigmoid, tanh       |
| Output dimension       | 10                        |
| Non-linear function    | softmax                   |
| Number of epochs       | 100                       |
| Batch size             | 32                        |
| Optimizer              | RMSprop                   |
| Loss function          | categorical cross-entropy |

### 3.3.1.1 The Tanh Activation

The *tanh* activation has been used as the nonlinearity of the first hidden layer. To improve the performance of the model, we search for an optimal learning parameter  $\eta$ . There is a small amount of fluctuation in the testing accuracy; however, all variants converge to above 98%. The general trend among all three  $\eta$  values is that LSTM1 and LSTM2 have the closest prediction to the standard LSTM. Then LSTM5 follows, and finally LSTM4 and LSTM3. As it is shown, setting  $\eta = 0.002$  results in test accuracy score of 98.60% in LSTM3 (i.e., the fastest model with least number of parameters) which is close to the best test score of the standard LSTM, i.e., 99.09%.

In training LSTM4a, LSTM5a & LSTM6, initially, there is a gap among different model variants. However, they all catch up with the base (standard) LSTM quickly within the 100 epochs. As one increases  $\eta$ , more fluctuation is observed. However, the models still sustain their performance levels. One advantage of the model variant LSTM6 is that it rises faster in comparison to other variants. However,  $\eta = 0.002$  causes it to decrease, indicating that this  $\eta$  value is relatively large. In all cases, LSTM4a and LSTM5a performance curves overlap. With a little offset, LSTM10 and LSTM11 also perform reasonably. The best results obtained among all the epochs are shown in Table 3.2.

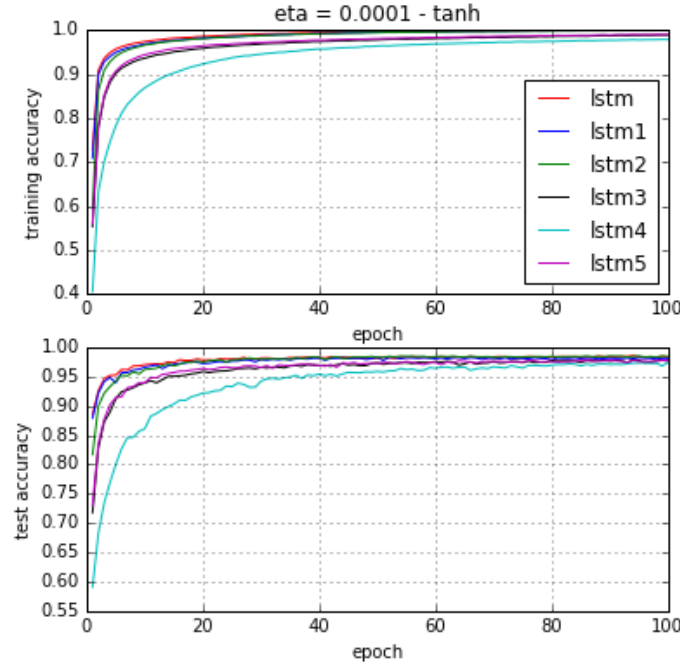


Figure 3.1: MNIST - Training & Test accuracy,  $\sigma = \tanh$ ,  $\eta = 1e-4$

### 3.3.1.2 The Sigmoid activation

Then, the sigmoid activation has been used as the nonlinearity of the first hidden layer. The same trend is observed using the sigmoid nonlinearity. In this case, one can clearly observe the training profile of each model. LSTM1, LSTM2, LSTM5, LSTM4, and LSTM3 have the closest prediction to the base LSTM, respectively. Again larger  $\eta$  results in better test accuracy and more fluctuation. It is observed that setting  $\eta = 0.002$  results in a test score of 98.34% in LSTM3, which is close to the test score of base LSTM 98.86%.

For the second group, the only difference is that the sigmoid activation with  $\eta = 1e-4$  slowly progresses towards its maximal performance. After performing parameter tuning, the typical test accuracy of the base LSTM model is 99%, test accuracy of variants LSTM4 and LSTM5 is about 98%, and test accuracy of variants LSTM4a and LSTM5a are 97%. As it is shown in the table and associated plots,  $\eta = 1e-4$  seems relatively small when using the *sigmoid* activation. As evidence, LSTM6 attains only a training and test accuracy of about 74% after 100 epochs, while

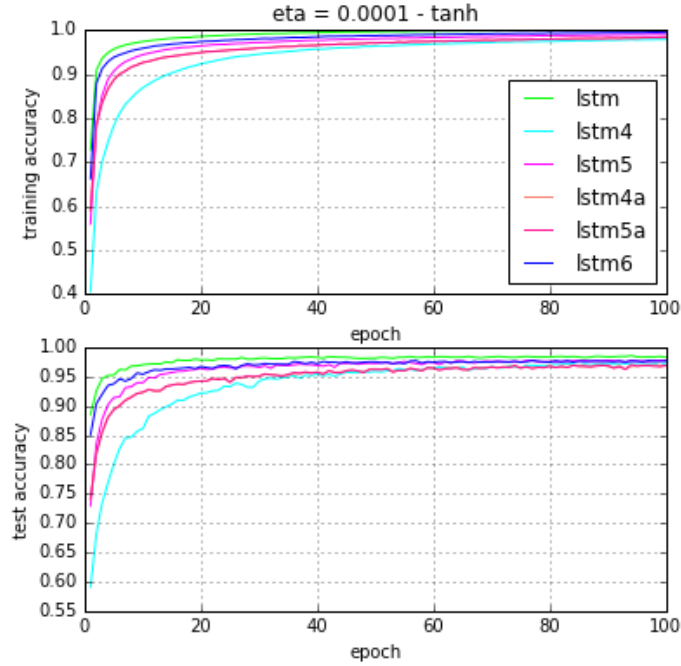


Figure 3.2: MNIST - Training & Test accuracy,  $\sigma = \tanh$ ,  $\eta = 1e-4$

it attains an accuracy of about 97% when  $\eta$  is increased 10-fold. In our variants, models using *tanh* saturate quickly; however, models using *sigmoid* rise steady from beginning to the last epoch. In these cases, again, LSTM4a and LSTM5a performances overlap. The third group also shows promising results. The best results obtained over the 100 epochs are summarized in Table 3.3.

### 3.3.1.3 The ReLU activation

Using *relu* activation as the nonlinearity of the first hidden layer, it is observed that the performance of LSTM, LSTM1, and LSTM2 drops after a number of epochs; however, this is not the case for LSTM3, LSTM4, and LSTM5. These latter models are sustained for all three choices of  $\eta$ . Also, LSTM3, the fastest model with the least number of parameters, shows the best performance among all five variants! With the *relu* as nonlinearity, the models fluctuate for larger  $\eta$ , which is not within the tolerance range of the model. Setting  $\eta = 0.002$  results in test score of 99.00% for LSTM3 which beat the best test score of the base LSTM, i.e., 98.43%. The best results obtained for all

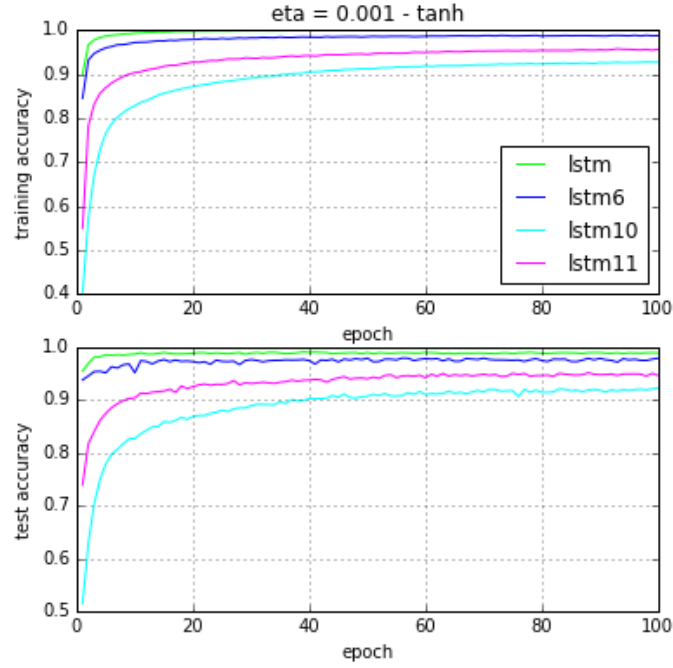


Figure 3.3: MNIST - Training & Test accuracy,  $\sigma = \tanh$ ,  $\eta = 1e-3$

models are summarized in table 3.4.

In the second group, all models perform well with  $\eta = 1e-4$ . In this case, LSTM4a and LSTM5a do not overlap. For  $\eta = 1e-3$ , the base LSTM does not sustain its performance and drastically drops. One may need to leverage an early stopping strategy to avoid this problem. In this case, the LSTM model begins to fall around epoch=50. LSTM6 also drops in performance. Model LSTM4, LSTM5, LSTM4a and LSTM5a show sustained accuracy performance as for  $\eta = 1e-4$ . For  $\eta = 2e-3$  model, LSTM4 and LSTM5 are still sustaining their performances. In the third group, LSTM6 and LSTM11 perform well, and LSTM10 does not rise.

We have explored a range of  $\eta$  for LSTM10 with *relu* activation to improve its performance. However, the effort was not successful. Increasing  $\eta$  from  $2e-6$  to  $1e-5$  leads to an increase in accuracy with value of 53.13%. For  $\eta$  less than  $1e-5$ , the plots have an increasing trend. However, after this point, the accuracy starts to drop after several epochs depending on the value of  $\eta$ .

Table 3.2: MNIST - Best results obtained using *tanh*.

|        |       | $\eta = 1e-4$ | $\eta = 1e-3$ | $\eta = 2e-3$ |
|--------|-------|---------------|---------------|---------------|
| LSTM   | train | 0.9995        | 1.0000        | 0.9994        |
|        | test  | 0.9853        | <b>0.9909</b> | 0.9903        |
| LSTM1  | train | 0.9993        | 0.9999        | 0.9996        |
|        | test  | 0.9828        | 0.9906        | <b>0.9907</b> |
| LSTM2  | train | 0.999         | 0.9997        | 0.9995        |
|        | test  | 0.9849        | 0.9897        | <b>0.9897</b> |
| LSTM3  | train | 0.9889        | 0.9977        | 0.9983        |
|        | test  | 0.9781        | 0.9827        | <b>0.9860</b> |
| LSTM4  | train | 0.9785        | 0.9975        | 0.9958        |
|        | test  | 0.9734        | <b>0.9853</b> | 0.9834        |
| LSTM5  | train | 0.9898        | 0.9985        | 0.9983        |
|        | test  | 0.9774        | 0.9835        | <b>0.9859</b> |
| LSTM4a | train | 0.9835        | 0.9957        | 0.9944        |
|        | test  | 0.9698        | <b>0.9803</b> | 0.9792        |
| LSTM5a | train | 0.9836        | 0.9977        | 0.998         |
|        | test  | 0.9700        | 0.9820        | <b>0.9821</b> |
| LSTM6  | train | 0.9948        | 0.9879        | 0.9657        |
|        | test  | 0.9771        | <b>0.9792</b> | 0.9656        |
| LSTM10 | train | -             | 0.9273        | -             |
|        | test  | -             | <b>0.9225</b> | -             |
| LSTM11 | train | -             | 0.9573        | -             |
|        | test  | -             | <b>0.9514</b> | -             |

### 3.3.2 IMDB

The IMDB Datasets is a binary sentiment classification dataset. In our model, a dictionary size of 5000 has been used. Each review is truncated or padded to 500 words. The first layer is an embedding layer which is a simple multiplication that transforms words into their corresponding word embedding. The output is then passed to an LSTM layer following a dense layer. We used the Adam optimizer, a batch size of 32, and the binary cross-entropy loss function. The network specification, adopted from Keras 1.2 examples, is given in table 3.5.

#### 3.3.2.1 The Sigmoid activation

In this case, also LSTM1, LSTM2, LSTM3, LSTM5, and LSTM4 have the closest prediction to the base LSTM, respectively. Setting  $\eta = 0.0001$  results in an almost fluctuation-free profile. It is

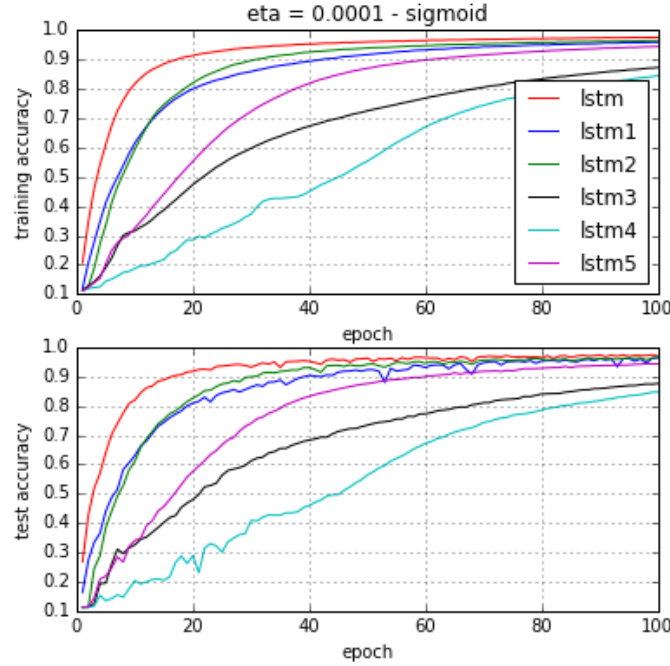


Figure 3.4: MNIST - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1e-4$

observed that setting  $\eta = 0.002$  results in a test score of 86.86% in LSTM3, which is close to the test score of base LSTM 88.68%.

LSTM5a, LSTM4a, LSTM5, LSTM4, and LSTM6 have the closest prediction to the base LSTM, respectively. In these cases, LSTM4a and LSTM5a performance curves overlap. It is observed that setting  $\eta = 0.002$  results in a test score of 89.12% in LSTM5a, which beat the test score of base LSTM, 88.68%.

LSTM11 and LSTM6 have the closest prediction to the base LSTM. Here, as we expected, LSTM11 shows better performance than LSTM10. Setting  $\eta = 0.0001$  results in a very smooth profile and less fluctuation. The best results obtained over the 100 epochs are summarized in Table 3.6.

To compensate for a decreased number of parameters, the dimension of hidden units has been increased along with having different, smaller values of  $\eta$  (figure 3.16). As it is shown, higher dimensions need fewer epochs to reach leveling off profiles. Setting  $\eta = 1.25e-5$ , creates an

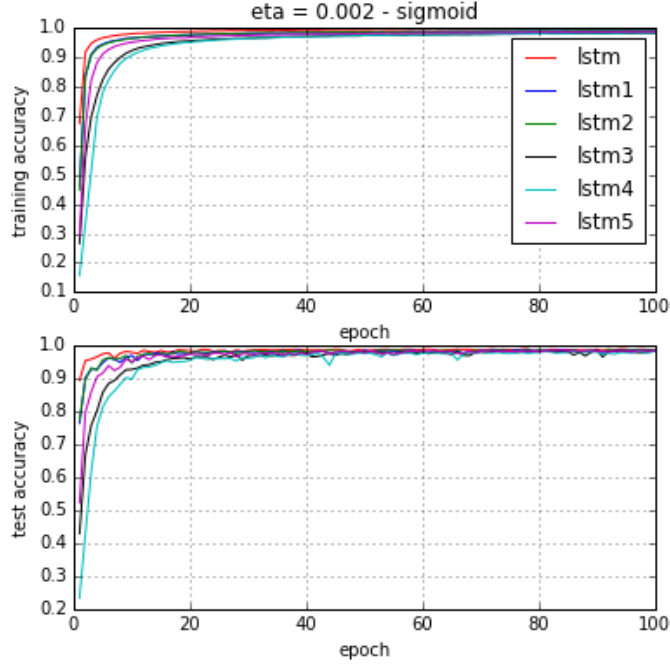


Figure 3.5: MNIST - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 2e-3$

almost fluctuation free profile. In this figure, 'lstm62' stands for LSTM6 using 200 hidden units.

The forget-gate constant value must be less than one in absolute value for bounded-input-bounded-output (BIBO) stability Salem (2016).  $f > 0.59$  did not work for the MNIST dataset and made the model unstable during training. We initially start with the same forget hyper-parameter  $f$  value on the IMDB dataset and then gradually increase it. We observe that the network retains BIBO stable performance up to  $f_t = 0.96$ . In figure 3.17, lstm629 denotes LSTM6 using  $h = 200$  and  $f = 0.99$ . Note that the accuracy plot profiles (figure 3.17 ) show an increasing trend and do not appear to level off after 100 epochs. We run the network training for 200 epochs; it is observed that LSTM6 surpasses *standard* LSTM at around epoch 150.

### 3.3.3 20 Newsgroups

The 20 Newsgroups dataset is a collection of 20,000 documents containing 20 different newsgroups. GloVe embedding is used to pre-train the model. The network architecture is adapted from Keras1.2



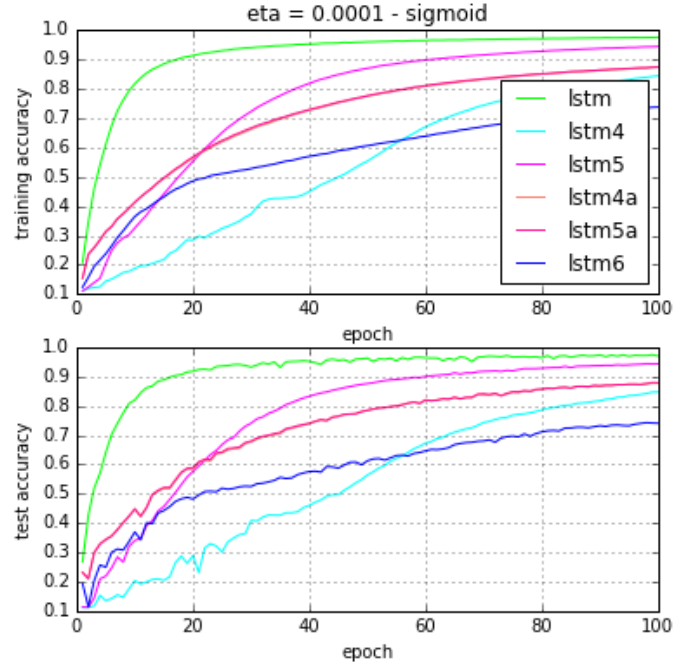


Figure 3.6: MNIST - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1e-4$

examples. We used the RMSprop optimizer, a batch size of 128, and the categorical cross-entropy loss function. Table 3.7 provides the network specification. We have applied our variants in the bidirectional layer.

For this dataset, we evaluate our new model LSTM6a as well. In LSTM6a, the matrix  $U_c$  in the cell equation is replaced with a corresponding vector  $u_c$ , in order to render a point-wise multiplication instead. Thus, the variant equations become

$$\begin{aligned}
 i_t &= 1.0 \\
 f_t &= f, \quad -1 < f < 1.0 \\
 o_t &= 1.0 \\
 \tilde{c}_t &= \sigma(W_c x_t + u_c \odot h_{t-1} + b_c) \\
 c_t &= f_t \odot c_{t-1} + i_t \odot \tilde{c}_t \\
 h_t &= o_t \odot \sigma(c_t)
 \end{aligned} \tag{3.13}$$

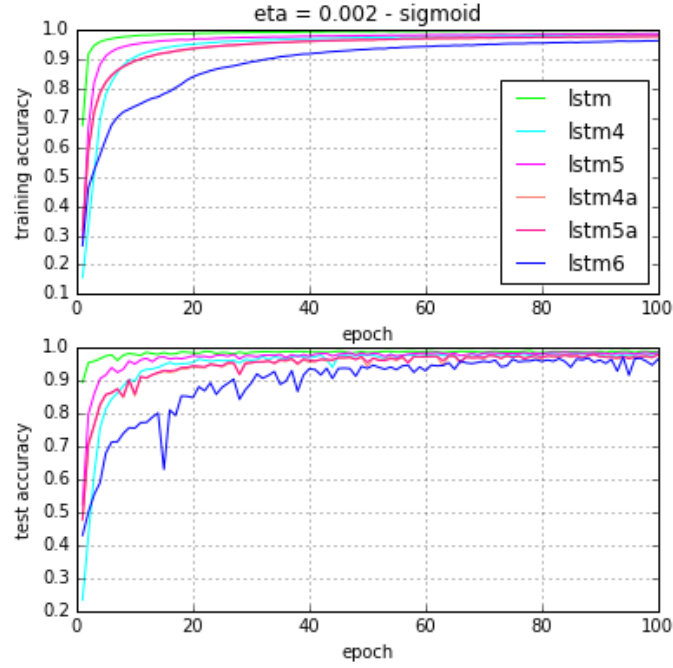


Figure 3.7: MNIST - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 2e-3$

### 3.3.3.1 The Tanh activation

Due to a considerable amount of fluctuation when using *sigmoid*, we decide to only depict results for *tanh* nonlinearity. As it is shown, LSTM1, LSTM2, LSTM3, and LSTM5 & LSTM6a show better performance than standard LSTM. Also, LSTM4a, LSTM5a & LSTM 10 have a very close prediction to it. As it is shown, setting  $\eta = 0.001$  results in a test accuracy score of 81.33% in LSTM3, which outperforms the best test score of the standard LSTM, i.e., 77.75%.

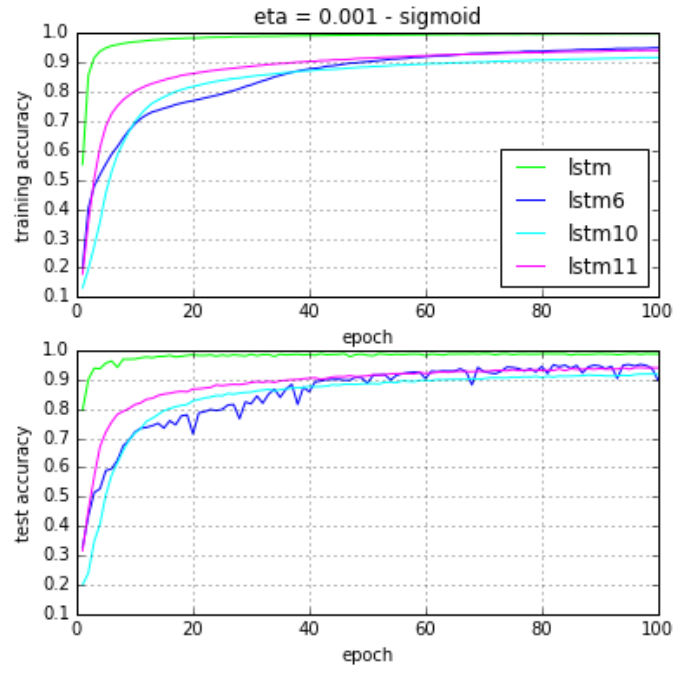


Figure 3.8: MNIST - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1e-3$

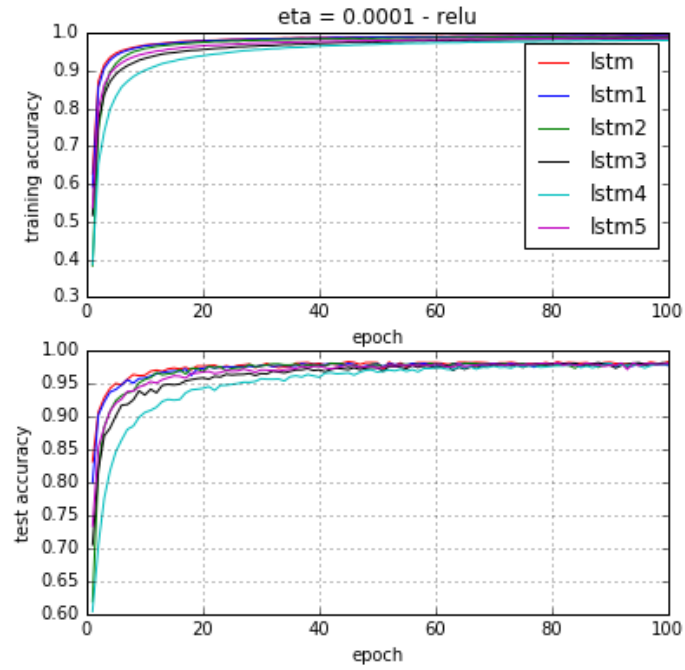


Figure 3.9: MNIST - Training & Test accuracy,  $\sigma = \text{relu}$ ,  $\eta = 1e-4$

Table 3.3: MNIST - Best results obtained using *sigmoid*.

|        |       | $\eta = 1e-4$ | $\eta = 1e-3$ | $\eta = 2e-3$ |
|--------|-------|---------------|---------------|---------------|
| LSTM   | train | 0.9751        | 0.9972        | 0.9978        |
|        | test  | 0.9739        | 0.9880        | <b>0.9886</b> |
| LSTM1  | train | 0.9584        | 0.9901        | 0.9905        |
|        | test  | 0.9635        | <b>0.9863</b> | 0.9858        |
| LSTM2  | train | 0.9636        | 0.9901        | 0.9907        |
|        | test  | 0.9660        | 0.9856        | <b>0.9858</b> |
| LSTM3  | train | 0.8721        | 0.9787        | 0.9828        |
|        | test  | 0.8757        | 0.9796        | <b>0.9834</b> |
| LSTM4  | train | 0.8439        | 0.9793        | 0.9839        |
|        | test  | 0.8466        | 0.9781        | <b>0.9822</b> |
| LSTM5  | train | 0.9438        | 0.9849        | 0.9879        |
|        | test  | 0.9431        | 0.9829        | <b>0.9844</b> |
| LSTM4a | train | 0.8728        | 0.9726        | 0.9778        |
|        | test  | 0.8770        | 0.9720        | <b>0.9768</b> |
| LSTM5a | train | 0.8742        | 0.9725        | 0.9789        |
|        | test  | 0.8788        | 0.9707        | <b>0.9783</b> |
| LSTM6  | train | 0.7373        | 0.9495        | 0.9636        |
|        | test  | 0.7423        | 0.9513        | <b>0.9700</b> |
| LSTM10 | train | -             | 0.9168        | -             |
|        | test  | -             | 0.9184        | -             |
| LSTM11 | train | -             | 0.9407        | -             |
|        | test  | -             | 0.9403        | -             |

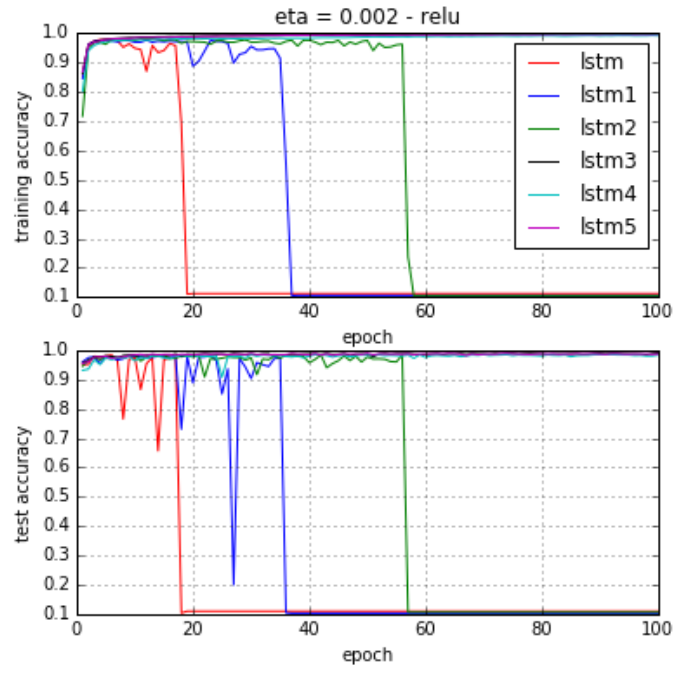


Figure 3.10: MNIST - Training & Test accuracy,  $\sigma = \text{relu}$ ,  $\eta = 2e-3$

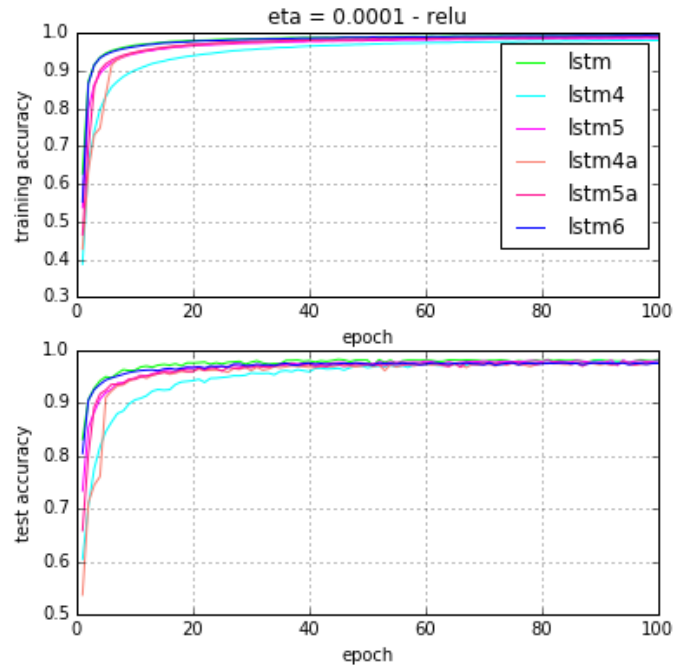


Figure 3.11: MNIST - Training & Test accuracy,  $\sigma = \text{relu}$ ,  $\eta = 1e-4$

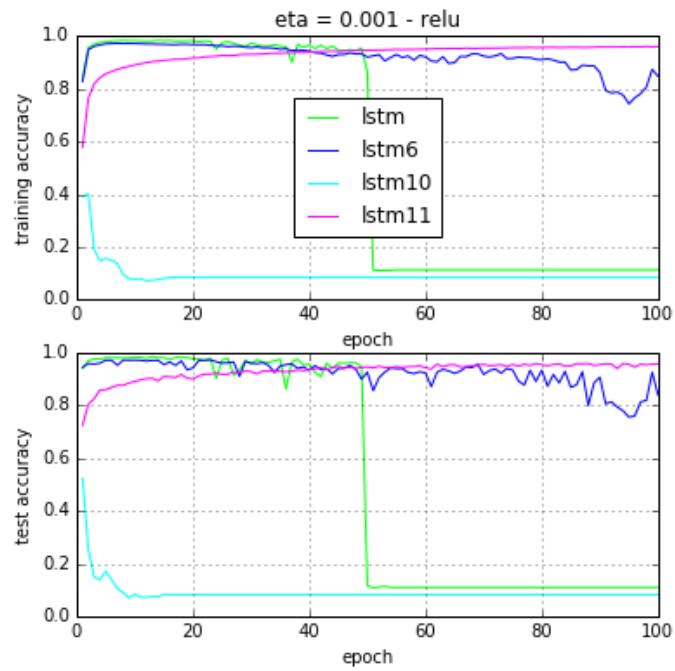


Figure 3.12: MNIST - Training & Test accuracy,  $\sigma = \text{relu}$ ,  $\eta = 1e-3$

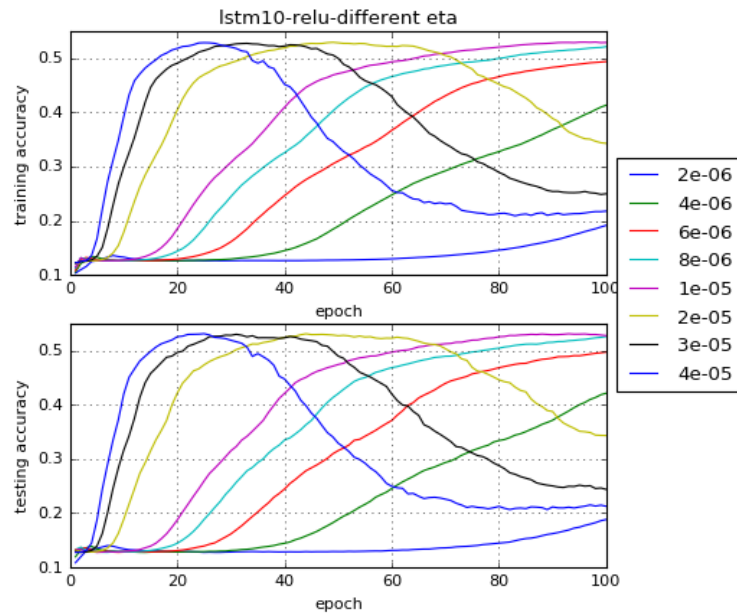


Figure 3.13: MNIST - Training & Test accuracy of different  $\eta$ , lstm10,  $\text{relu}$

Table 3.4: MNIST - Best results obtained using *relu*.

|        |       | $\eta = 1e-4$ | $\eta = 1e-3$ | $\eta = 2e-3$ |
|--------|-------|---------------|---------------|---------------|
| LSTM   | train | 0.9932        | 0.9829        | 0.9787        |
|        | test  | 0.9824        | <b>0.9843</b> | 0.9833        |
| LSTM1  | train | 0.9926        | 0.9824        | 0.9758        |
|        | test  | 0.9803        | <b>0.9832</b> | 0.9806        |
| LSTM2  | train | 0.9896        | 0.9795        | 0.98          |
|        | test  | 0.9802        | 0.9805        | <b>0.9836</b> |
| LSTM3  | train | 0.9865        | 0.9967        | 0.9968        |
|        | test  | 0.9808        | 0.9882        | <b>0.9900</b> |
| LSTM4  | train | 0.9808        | 0.9916        | 0.9918        |
|        | test  | 0.9796        | <b>0.9857</b> | 0.9847        |
| LSTM5  | train | 0.987         | 0.9962        | 0.9964        |
|        | test  | 0.9807        | 0.9885        | <b>0.9892</b> |
| LSTM4a | train | 0.9906        | 0.9949        | 0.1124        |
|        | test  | 0.9775        | <b>0.9878</b> | 0.1135        |
| LSTM5a | train | 0.9904        | 0.996         | 0.1124        |
|        | test  | 0.9769        | <b>0.9856</b> | 0.1135        |
| LSTM6  | train | 0.9935        | 0.9719        | 0.09737       |
|        | test  | <b>0.9761</b> | 0.9720        | 0.0982        |
| LSTM10 | train | -             | 0.4018        | -             |
|        | test  | -             | 0.5226        | -             |
| LSTM11 | train | -             | 0.9597        | -             |
|        | test  | -             | 0.9582        | -             |

Table 3.5: IMDB - Network specifications.

|                        |                     |
|------------------------|---------------------|
| Input dimension        | $32 \times 500$     |
| Number of hidden units | 100                 |
| Non-linear function    | tanh, sigmoid, tanh |
| Output dimension       | 2                   |

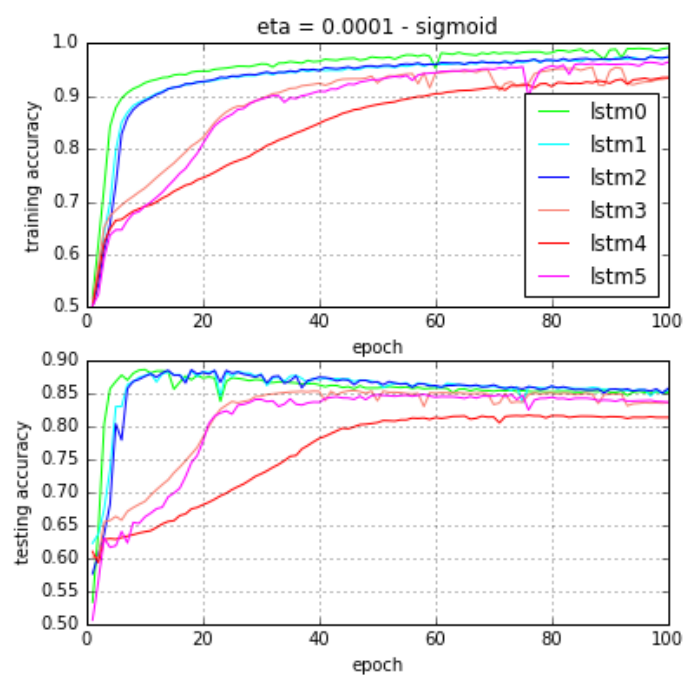


Figure 3.14: IMDB - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1e-4$

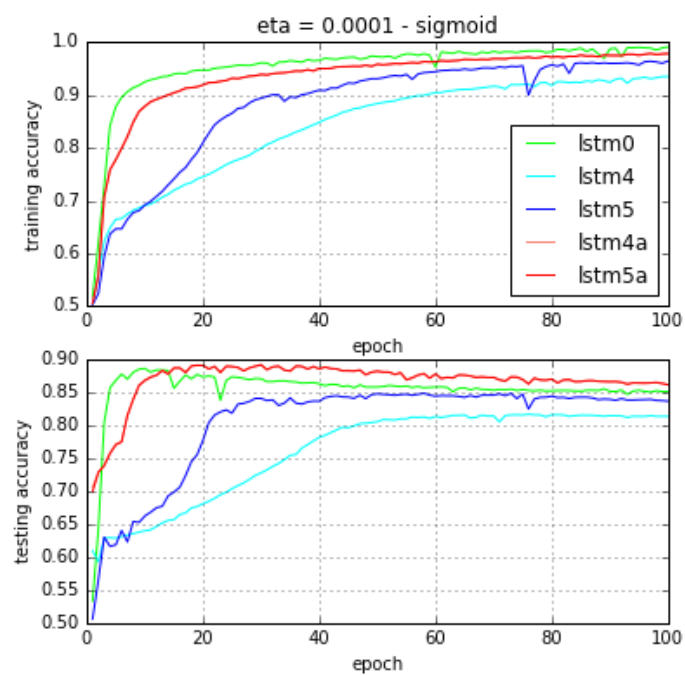


Figure 3.15: IMDB - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1e-4$



Table 3.6: IMDB - Best results obtained sigmoid.

|        |       | $\eta = 1e-4$ | $\eta = 1e-3$ | $\eta = 2e-3$ |
|--------|-------|---------------|---------------|---------------|
| LSTM   | train | 0.9906        | 1.0000        | 1.0000        |
|        | test  | 0.8856        | <b>0.8868</b> | 0.8824        |
| LSTM1  | train | 0.9744        | 0.9923        | 0.9968        |
|        | test  | 0.8835        | <b>0.8840</b> | 0.8766        |
| LSTM2  | train | 0.9746        | 0.9929        | 0.9954        |
|        | test  | <b>0.8853</b> | 0.8804        | 0.8800        |
| LSTM3  | train | 0.9542        | 0.9863        | 0.9955        |
|        | test  | 0.8537        | 0.8426        | <b>0.8686</b> |
| LSTM4  | train | 0.9346        | 0.9616        | 0.9920        |
|        | test  | 0.8165        | 0.8204        | <b>0.8624</b> |
| LSTM5  | train | 0.9637        | 0.9934        | 0.9982        |
|        | test  | 0.8482        | 0.8702        | <b>0.8746</b> |
| LSTM4a | train | 0.9781        | 0.9897        | 0.9955        |
|        | test  | <b>0.8914</b> | 0.8790        | 0.8766        |
| LSTM5a | train | 0.9782        | 0.9901        | 0.9955        |
|        | test  | <b>0.8912</b> | 0.8790        | 0.8762        |
| LSTM10 | train | 0.7993        | 0.9989        | 0.9909        |
|        | test  | 0.7112        | <b>0.8512</b> | 0.8470        |
| LSTM11 | train | 0.9104        | 0.9996        | 0.9969        |
|        | test  | 0.8374        | 0.8564        | <b>0.8596</b> |

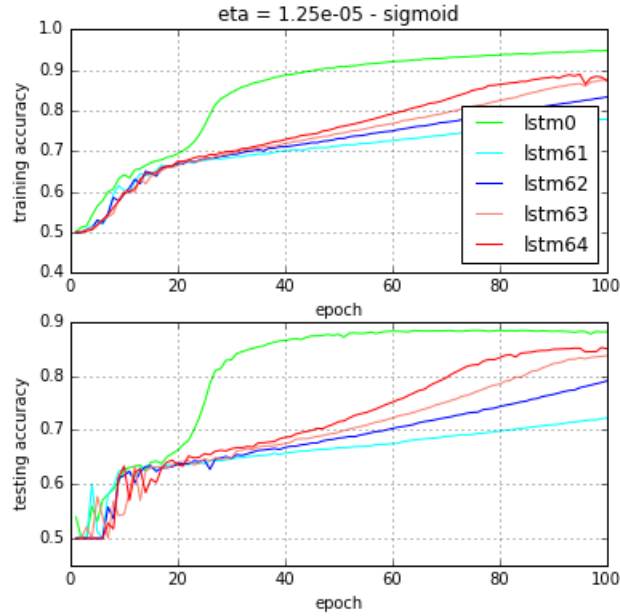


Figure 3.16: IMDB - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1.25e-5$

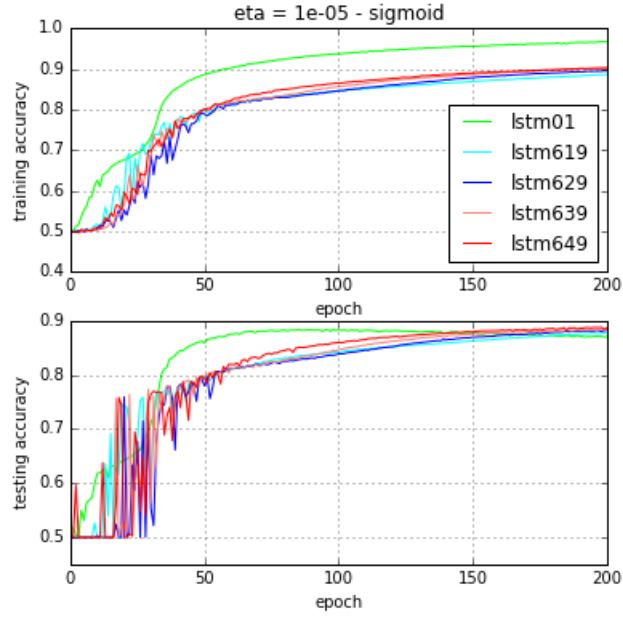


Figure 3.17: IMDB - Training & Test accuracy,  $\sigma = \text{sigmoid}$ ,  $\eta = 1e-5$

Table 3.7: News20 - Network specifications.

|                        |                   |
|------------------------|-------------------|
| Input dimension        | 1000              |
| Embedding layer        | $1000 \times 100$ |
| Conv1D(128, 5, 'relu') | $996 \times 128$  |
| Maxpooling1D(5)        | $199 \times 128$  |
| Conv1D(128, 5, 'relu') | $195 \times 128$  |
| Maxpooling1D(5)        | $39 \times 128$   |
| Conv1D(128, 5, 'relu') | $35 \times 128$   |
| Maxpooling1D(2)        | $17 \times 128$   |
| Number of epochs       | 100               |
| Bidirectional(lstm)    | 256               |
| Dense                  | 128               |
| Dense                  | 6                 |

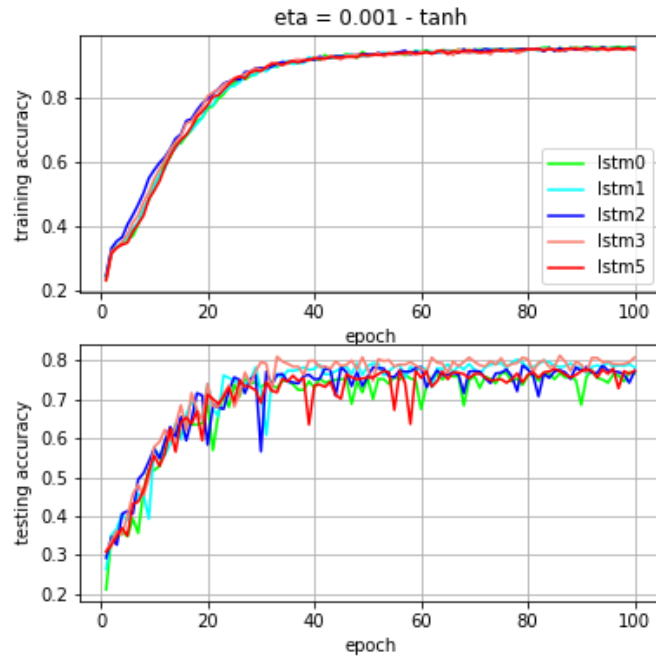


Figure 3.18: News20 - Training & Test accuracy,  $\sigma = \tanh$ ,  $\eta = 1e-3$

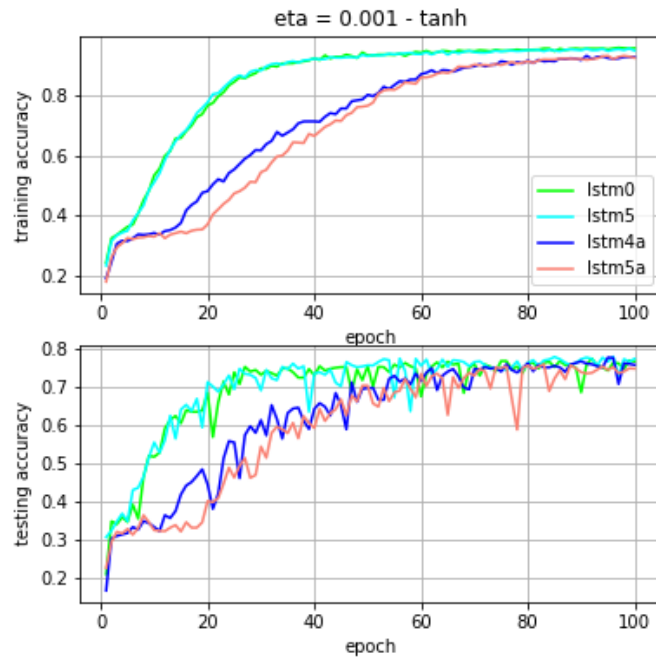


Figure 3.19: News20 - Training & Test accuracy,  $\sigma = \tanh$ ,  $\eta = 1e-3$

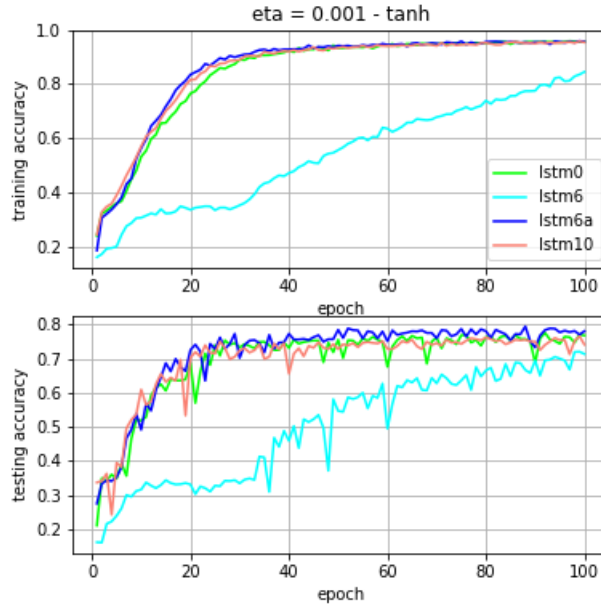


Figure 3.20: News20 - Training & Test accuracy,  $\sigma = \tanh$ ,  $\eta = 1e-3$

Table 3.8: News20 - Best results obtained using  $\tanh$ .

|        |       | $\eta = 5e-4$ | $\eta = 1e-3$ | $\eta = 2e-3$ |
|--------|-------|---------------|---------------|---------------|
| LSTM   | train | 0.9519        | 0.9581        | 0.9600        |
|        | test  | 0.7592        | 0.7750        | <b>0.7775</b> |
| LSTM1  | train | 0.9508        | 0.9554        | 0.9563        |
|        | test  | 0.7608        | 0.8033        | <b>0.8058</b> |
| LSTM2  | train | 0.9415        | 0.9560        | 0.9552        |
|        | test  | 0.7517        | 0.7883        | <b>0.7942</b> |
| LSTM3  | train | 0.9513        | 0.9563        | 0.9567        |
|        | test  | 0.7917        | <b>0.8133</b> | 0.7925        |
| LSTM5  | train | 0.9519        | 0.9552        | 0.9575        |
|        | test  | 0.7783        | 0.7800        | <b>0.7967</b> |
| LSTM4a | train | 0.9181        | 0.9313        | 0.9163        |
|        | test  | 0.7558        | <b>0.7775</b> | 0.7708        |
| LSTM5a | train | 0.9271        | 0.9313        | 0.9254        |
|        | test  | 0.7558        | 0.7600        | <b>0.7700</b> |
| LSTM6  | train | 0.8169        | 0.8448        | 0.7850        |
|        | test  | 0.7158        | <b>0.7200</b> | 0.7100        |
| LSTM6a | train | 0.9552        | 0.9583        | 0.9556        |
|        | test  | 0.7792        | <b>0.7942</b> | 0.7842        |
| LSTM10 | train | 0.9506        | 0.9571        | 0.9573        |
|        | test  | 0.7250        | 0.7650        | <b>0.7758</b> |

### **3.4 Conclusion**

Nine variants of the base LSTM model have been presented and evaluated. These models have been examined and evaluated using different nonlinearity and different learning rates on the three classical datasets. It has been found that new model variants are comparable to the standard LSTM model. Thus, these variant models may be suitably chosen in applications in order to benefit from the speed and/or computational cost.

## CHAPTER 4

### FEATURE EXTRACTOIN MODEL

Early SLR systems performed data acquisition using sensor-based devices such as data gloves and accelerometers. These devices provide position, orientation, velocity, and other specifics of the hands. However, due to the prohibitive costs of such approaches, vision-based devices have been introduced. Microsoft Kinect, Leap Motion Controller, and Google Tango provide RGB and depth-maps information, which can be used as an image-based input to a network. In one of our solutions to the SLR task in the next chapters, we go one step further and propose to acquire data using a regular camera and then extract the key characteristics of each sign using MediaPipe Hands and Pose modules.

MediaPipe is a graph-based framework for building multimodal (video, audio, and sensor) applied machine learning pipelines MediaPipe (2019) introduced by Google. Mediapipe's use of graphs and calculators means that the work of one project can easily translate to the work of another. One advantage of MediaPipe is that it does not rely on powerful desktop environments for inference and achieves real-time performance on mobile phones, laptops, and even on the web. In addition, by employing GPU acceleration and multi-threading, MediaPipe is one of the fastest ML solutions available.

#### 4.1 MediaPipe Hands

Detecting hands is a complex task. The model has to recognize a variety of hand sizes and also address occluded hands issues. In addition, as there are not many distinguishing features in hands themselves, additional context like arm and body are required to locate the hands accurately. MediaPipe hands module addresses these challenges by first training a palm detector and then training a hand landmark model on the generated cropped hand images. The cropped images can also be generated based on the hand landmarks identified in the previous frame. The palm detection model is invoked only when the landmark model can no longer identify hand presence.

The palm detection model operates over the whole image and returns cropped hand images. Mediapipe proposes to employ a palm detection model instead of a hand detection one since estimating bounding boxes of rigid objects like palms is significantly simpler than detecting hands with fingers. Also, non-maximum suppression algorithms work well on small objects like palms. Moreover, since palms can be modeled using square bounding boxes, other aspect ratios can be ignored. An average precision of 95.7% in the palm detection model has been reported.

The hand landmark model performs keypoint localization of 21 3D coordinates on the cropped hand images generated by the palm detection model using regression (see figure 4.1). To perform supervised training, they have manually annotated 30K real-world images with 21 3D coordinates. They have also generated a synthetic hand model over various backgrounds and map it to the corresponding 3D coordinates to provide additional supervision.

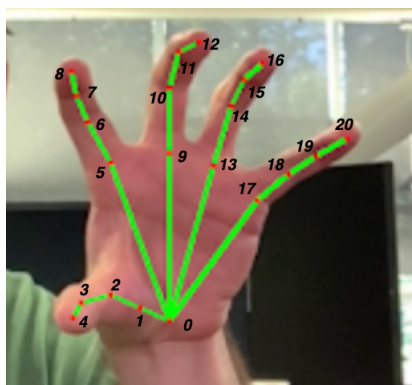


Figure 4.1: MediaPipe hands tracking module output example

## 4.2 MediaPipe Pose

MediaPipe pose module infers 25 2D upper-body landmarks by first locating the pose region-of-interest (ROI) and then training a pose landmark model on the generated region. The ROI can also be generated based on the pose landmarks identified in the previous frame. The pose detection model is invoked only when the landmark model can no longer identify body pose presence (see figure 4.2).

Inspired by Leonardo's Vitruvian man Vitruvian Man (1490), the pose detection model predicts

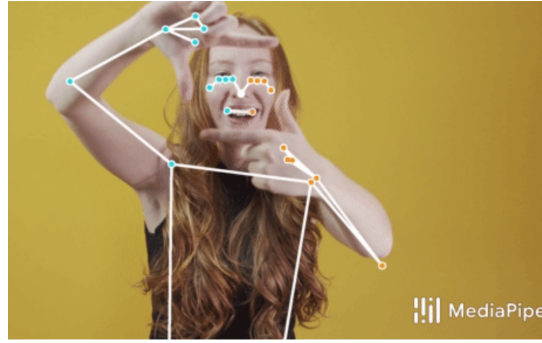


Figure 4.2: MediaPipe pose module output example MediaPipe (2019)

two virtual key-points, namely shoulder, and hip midpoints, that describe the circle circumscribing the person (see figure 4.3). The incline angle of the line connecting the shoulder and hip midpoints is also inferred. Similar to the hand landmark model, the pose landmark model performs keypoint localization of 25 upper-body coordinates on pose region-of-interest generated by the pose detection model.

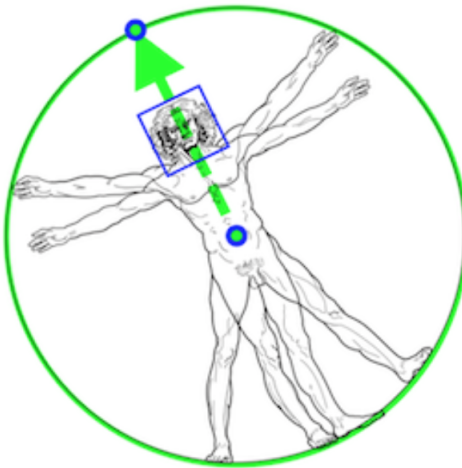


Figure 4.3: The proportions of the human body according to Vitruvius MediaPipe (2019)



## **CHAPTER 5**

### **CHARACTER-LEVEL SLR**

To investigate character-level Sign Language Recognition, we study two different datasets with uniform and complex backgrounds. To examine grayscale Sign Language MNIST, we propose an 8-layers Convolutional neural network. To study ASL Fingerspelling A datasets, we propose two different approaches. In the first approach, we use VGG19 pre-trained with ImageNet and fine-tune it. In the second approach, we use MediaPipe to extract 2D landmarks of the hand, and then create new black and white images with uniform background and then feed them to the same 8-layer architecture proposed for Sign Language MNIST. The experiment results demonstrate the effectiveness of our approach on both datasets. We achieve 99.9% accuracy for the first dataset and 96% & 98% accuracy for the second dataset. The implementation of the models is publicly accessible through Akandeh (2021).

#### **5.1 Introduction**

To perform a thorough study of Sign Language Recognition, we start by experimenting with the character-level SLR. A practical application for Character-level SLR is fingerspelling. Fingerspelling is used for the letter by letter signing of names, place names, age, numbers, date, year, and words that do not have pre-defined signs in the vocabulary Adithya & Rajesh (2020).

In this chapter, we study image-based input to the SLR system. We can then apply our findings in word-level and sentence-level SLR, where we have to deal with video-based input. First, we experiment with Sign Language MNIST, a grayscale-based dataset with uniform background. Then we investigate ASL Fingerspelling A datasets, in which hands are presented in complex background conditions. To deal with complex background dataset, we use domain knowledge to transform the raw images into the most discriminative representation by which the model can detect and differentiate the patterns accurately. We also try a second approach in which we minimize prior knowledge and feed raw images to the network.

### 5.1.1 Characteristics

Sign Languages use the hand gesture, facial expression and body pose to convey information. There are two types of static and dynamic hand gestures. Static hand gestures are formed by various shapes and orientations of hands without any motion involved. Dynamic hand gestures are formed by a sequence of hand postures with associated motion information Adithya & Rajesh (2020). In American sign language, all letters except "j" and "z" are represented by static hand gestures. Figure 5.1 illustrates ASL fingerspelling alphabet.



Figure 5.1: American Sign Language Alphabet Sign Language Club (2012)

### 5.1.2 Challenges

Among character-level, word-level, and sentence-level SLR, character-level is the easiest to deal with since it does not involve any motion, and inputs are in the form of images rather than videos. Also, American fingerspelling is single-handed, which removes the difficulties of hands occluding

one another. Still, character-level SLR is a challenging problem. Lighting conditions, complex background, signee body postures, and camera position can pose significant challenges on the task. Also, there are only subtle differences between different letters. For example, letters a, e, m, n, s, and t are all represented by a closed fist and differ only by the thumb position Pugeault & Bowden (2011). The thumb itself is also barely visible in m and n (figure 5.1). Moreover, signees can have their signing styles which leads to variability in the dataset.

### **5.1.3 Related Work**

Rioux-Maldague & Giguere (2014) applied Deep Belief Network on the American Sign Language fingerspelling dataset. They leveraged depth and intensity images obtained from the Microsoft Kinect sensor. The network consists of 3 Restricted Boltzmann Machines with the size of 1500, 700, and 400 units and one translation layer. Their architecture was capable of real-time sign classification, and it was also adaptive to any environment or lighting intensity. They were able to obtain 99% accuracy on multi-user with all known users and 77% accuracy on multi-user with unseen users.

Li et al. (2015) performed feature learning using a sparse autoencoder (SAE) and principal component analysis on ASL fingerspelling by Pugeault & Bowden (2011). An auto-encoder is a type of feed-forward neural network, under the unsupervised setting, whose output is required to be equal to the input. First, they performed dimension reduction on RGB and depth images using PCA and fed the results into SAE with CNNs. Then, the learned features from both channels have been dimensionality reduced further via PCA and then are concatenated and fed into a softmax classifier. The proposed feature-learning model obtained a recognition rate of 99.05%, outperforming state-of-the-art models at the time.

Huang et al. (2015) also adopted a three-layer Deep Belief Network, consisting of 500, 500, and 2000 hidden units. They collected two datasets containing 26 alphabet signs using Real-Sense and Kinect. Real-Sense is a camera device that can detect and track the location of hands. They recorded video clips from 3 different signers. Then, they extracted frames from recorded video

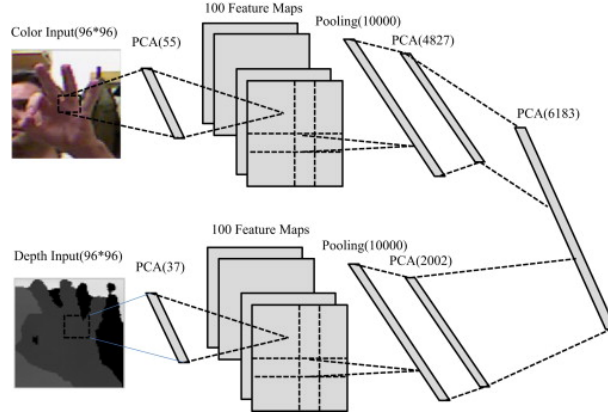


Figure 5.2: The character-level sign recognizer architecture proposed by Li et al. (2015)

clips and collected 65,000 frame images in total. Each image is then converted to a 66-dimensional feature vector, including the 3D coordinates of 22 finger points. For the Kinect dataset, they produced the same amount of data. However, Kinect does not support a hands or fingers model. Instead, they made use of color and depth information and then segmented hand-shape from the background. The resolution of hand-shape images was  $32 \times 32$ . Real-Sense and Kinect DNN were able to achieve 97.8% and 98.9% accuracy, respectively.

Aly et al. (2016) used an architecture called PCANet to extract local features from depth and intensity images using an unsupervised deep learning method on Arabic SL fingerspelling. The learned features are then fed into a linear support vector machine classifier. They were able to obtain an average accuracy of 99.5%.

Oyedotun & Khashman (2017) also adopted three different sized convolutional neural networks and three stacked denoising autoencoders (SDAEs) on 24 hand gestures of the Thomas Moeslunds gesture database. They studied CNNs with different depth sizes of 2, 3, and 4 and SDAEs with 1 to 4 hidden layers.

## 5.2 Proposed Models

To investigate character-level Sign Language Recognition, we study two different datasets with uniform and complex backgrounds. The Sign Language MNIST, which is patterned to match with the classic MNIST, includes images with uniform background, and the ASL Fingerspelling A

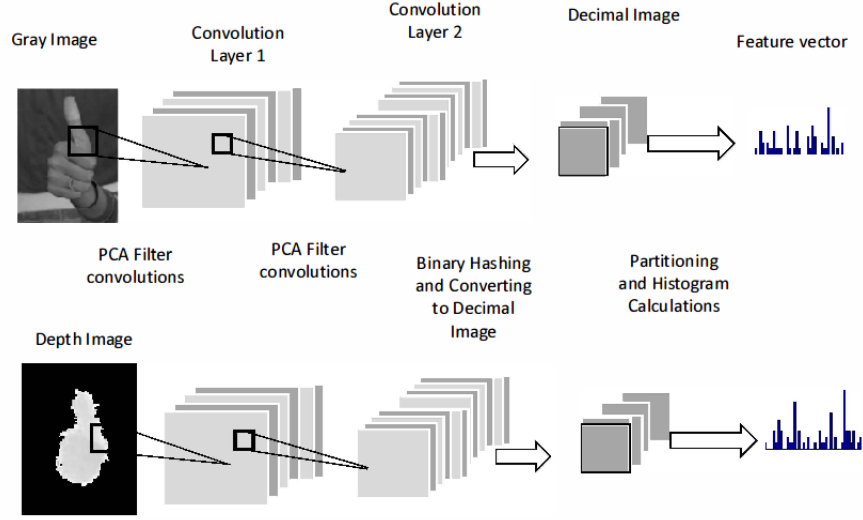


Figure 5.3: Feature extraction using PCANet Aly et al. (2016)

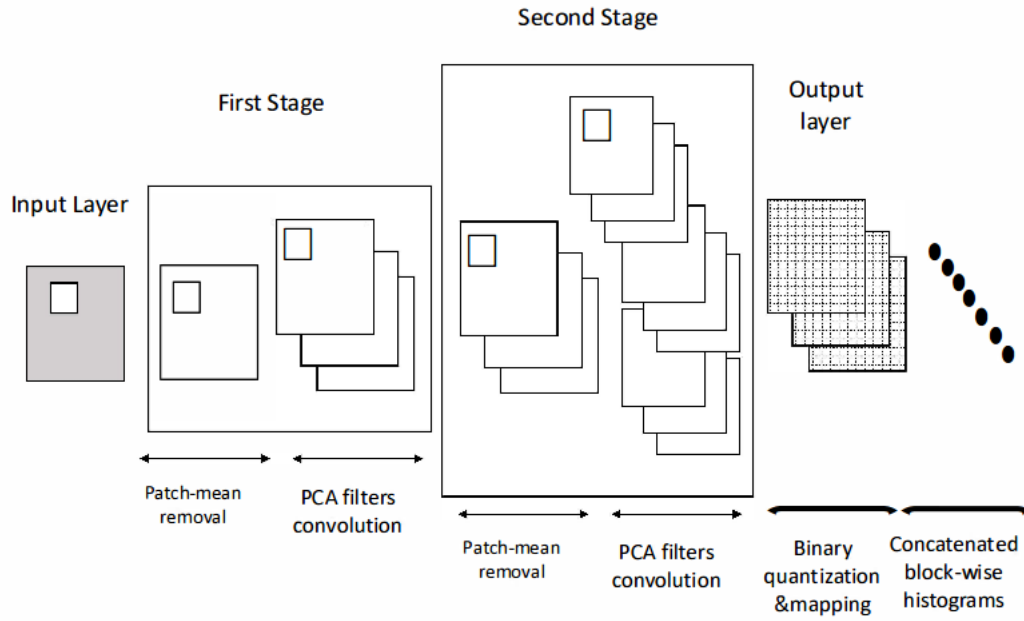


Figure 5.4: The character-level sign recognizer architecture proposed by Aly et al. (2016)

dataset has images with complex background. To examine grayscale Sign Language MNIST, we propose an 8-layers Convolutional neural network. To study ASL Fingerspelling A datasets, we propose two different approaches, one with using prior knowledge and one without.

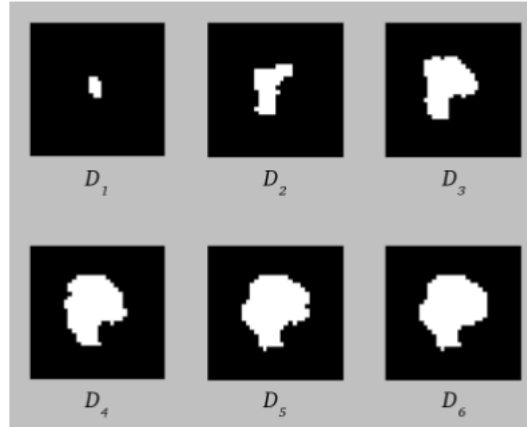


Figure 5.5: Successive binary depth images character 'G' Rioux-Maldague & Giguere (2014)

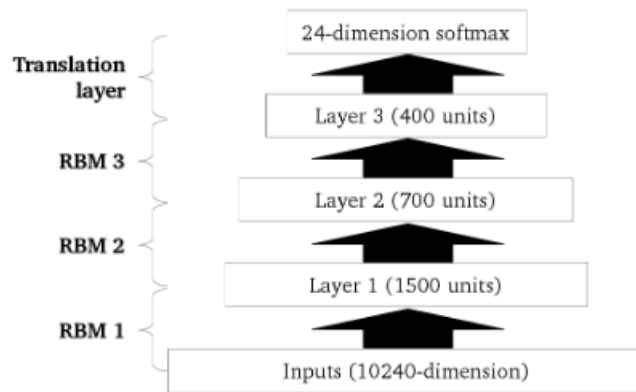


Figure 5.6: The character-level sign recognizer architecture proposed by Rioux-Maldague & Giguere (2014)

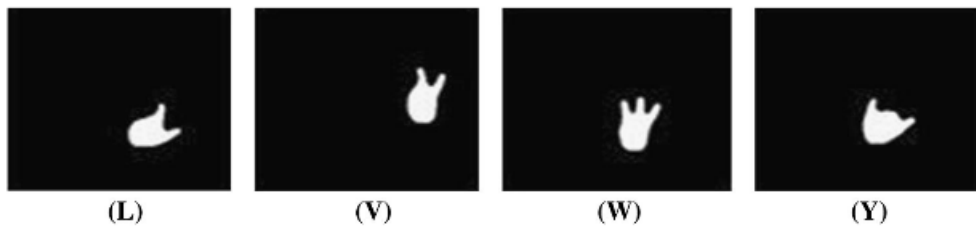


Figure 5.7: Unsegmented hand gesture samples Oyedotun & Khashman (2017)

### 5.2.1 Model Overview

To analyze and infer on Sign Language MNIST, we proposed an eight-layer convolutional network. The first six are alternating convolutional and max-pooling layers, and the last two are fully-



Figure 5.8: Finger joints capture by Real-Sense Huang et al. (2015)

connected layers. The first convolutional layer filters the  $28 \times 28$  input image with 128 kernels of size  $5 \times 5$  with a stride of 1. The second convolutional layer takes as input the response-normalized and pooled output of the first convolutional layer and filters it with 64 kernels of size  $2 \times 2$ . The third convolutional layer has 32 kernels of size  $2 \times 2$ , and finally, the fully-connected layers have 512 and 24 neurons, respectively.

To examine the ASL Fingerspelling A dataset, we propose two different approaches. In the first approach, raw data is fed to VGG-19 pre-trained with ImageNet dataset Deng et al. (2009) and the model is fine-tuned. VGG-19 is a convolutional neural network that is 19 layers deep Simonyan & Zisserman (2015). In the second approach, we create new black and white skeleton images (see figure 5.9) from extracted 2D landmarks of the hands using MediaPipe and feed them to the same network proposed for the Sign Language MNIST.

### 5.2.2 Model Details

A detailed description of the proposed model to train *Sign Language MNIST* and *ASL Fingerspelling A Skeleton* has been listed in table 5.1. To train *ASL Fingerspelling A raw data* we perform transfer learning. To fine-tune VGG-19 (figure 5.10) we discard the fully-connected layer heads and add our classifier layers of size 512, 512 & 24. We call the first proposed model 'SConv' and the second model 'SVGG' for the sake of easy referring.

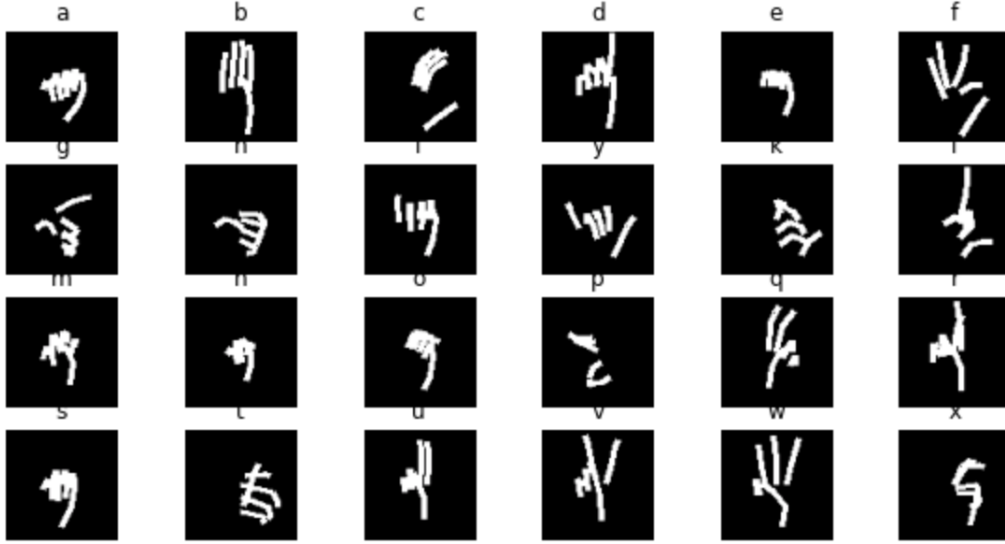


Figure 5.9: ASL alphabet skeleton created from 2D coordinates of the hands using MediaPipe

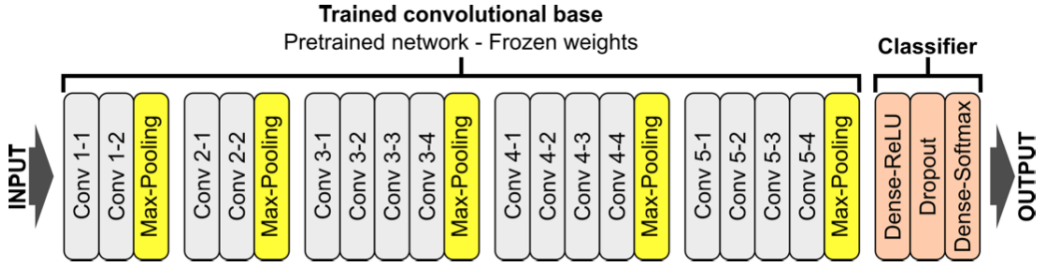


Figure 5.10: VGG-19 architecture Jaworek-Korjakowska et al. (2019)

### 5.3 Dataset

The Sign Language MNIST multi-signer Sign Language MNIST (2018), is a dataset for character-level sign language recognition created from the American Sign Language letter database Sign Language and Static gesture (2018) and patterned to match closely with the classic MNIST. This dataset contains 27455 images for training and 7172 images for testing. Each image is  $28 \times 28$  with grayscale value and includes hands-only. Sign Language MNIST represents a multi-class problem with 24 classes of letters (excluding J and Z, which require motion)

The second dataset is the American fingerspelling A dataset which contains 24 letters of the ASL alphabet excluding the letters “j” and “z” Rastgoo et al. (2018). The images of this set are



Table 5.1: Network used to train Sign Language MNIST and ASL Fingerspelling A Skeleton

|                       |                           |
|-----------------------|---------------------------|
| Input dimension       | $28 \times 28$            |
| Conv2D(128, 5,'relu') | $28 \times 28 \times 128$ |
| Maxpooling2D(2)       | $14 \times 14 \times 128$ |
| Conv2D(64, 2,'relu')  | $14 \times 14 \times 64$  |
| Maxpooling2D(2)       | $7 \times 7 \times 64$    |
| Conv2D(32, 2,'relu')  | $7 \times 7 \times 32$    |
| Maxpooling2D(2)       | $3 \times 3 \times 32$    |
| Flatten               | 288                       |
| Dense ('relu')        | 512                       |
| Dense ('softmax')     | 24                        |

captured in five different sessions, with five different users in similar lighting conditions in the presence of complex background objects.

## 5.4 Experimental Setup

MediaPipe requires packets (any data structure with a timestamp) to be fed into the graph in order to work. To make use of MediaPipe, we used FFmpeg to create videos from images in each class. FFmpeg is an open-source project consisting of a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. We then used generated 2D landmarks of the hands to create new black and white skeleton images. The skeleton images are constructed such that the center of the hand (point 9 in figure 4.1) is centered in the middle of the image, that is, both vertically and horizontally centered.

## 5.5 Experimental Results

The proposed models are evaluated by accuracy and confusion matrix. The confusion matrix is a specific table layout that allows visualization of the performance of the classification model by class. SConv achieved 99.9% accuracy on Sign Language MNIST for seen signers (figures 5.11). There is almost no confusion between similar-looking hand-shapes. The overall performance of SConv and SVGG models trained on *ASL Fingerspelling A Skeleton* for seen users across 7 participants are recorded in figures 5.12.

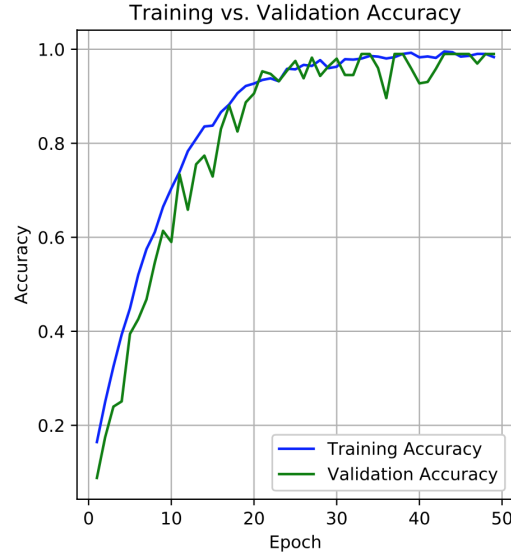


Figure 5.11: Training & Validation Accuracy for the SConv model on Sign Language MNIST dataset.

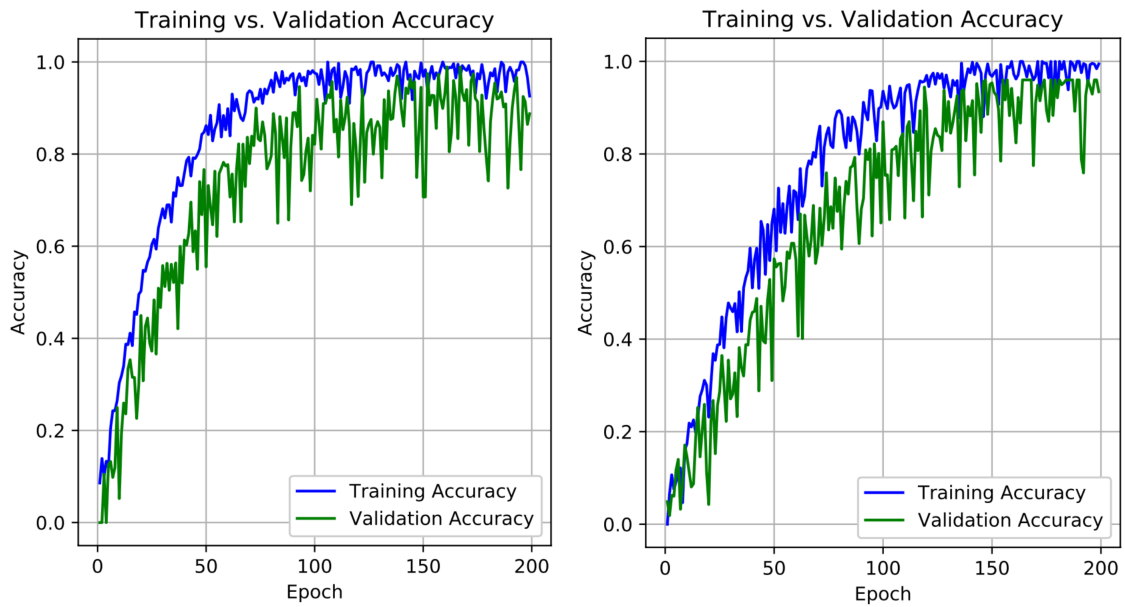


Figure 5.12: Training & Validation Accuracy of theSVGG (left) & SConv (right) models on Sign Language MNIST dataset.

Confusion between similar-looking hand-shapes, such as 'm' and 'n', and also 's', 't' and 'e' are noticeable in the confusion matrices (figures5.13). This is expected as these letters are signed as a fist that only differs by the thumb position, which is barely visible in some cases.

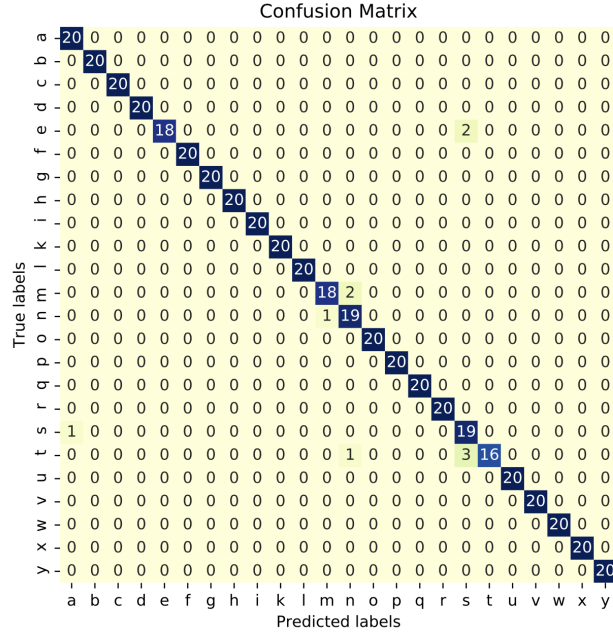


Figure 5.13: Confusion Matrix of SConv model on ASL Fingerspelling A datasets.

## 5.6 Discussion

With our presented novel model, we are able to classify 24 characters with 98% accuracy. By creating black and white skeleton images using Mediapipe from raw data, we overcome lighting conditions, complex background, and camera position challenges. Experimental results indicate that our model achieves substantial improvements over mainstream methods in terms of training speed. The SConv model starts to converge at 100 epochs, and SVGG model starts to converge at 150 epochs (figure 5.12). Given its promising performance, as there are approximately 40 common handshapes used in ASL, including the 24 static letters, we can use this model along with its weights as a base for the word-level classification task.

## 5.7 Summary

In this chapter, we experimented with different datasets with uniform and complex backgrounds. To examine grayscale Sign Language MNIST, we proposed an 8-layers Convolutional neural network. To study ASL Fingerspelling A datasets, we investigated two different approaches. In the

first approach, we use VGG19 pre-trained with ImageNet and fine-tune it. In the second approach, we use MediaPipe to extract 2D landmarks of the hand, and then create the skeleton of the hands with uniform background and then feed them to the same 8-layer architecture proposed for Sign Language MNIST. The experiment results demonstrated the effectiveness of our approach on both datasets. We achieve 99.9% accuracy for the first dataset and 96% & 98% accuracy for the second dataset.

## CHAPTER 6

### WORD-LEVEL SLR

To investigate word-level SLR, we present two deep dynamic sign language recognition frameworks. In the first framework, to minimize prior knowledge, we feed raw frames into an 18-layers LRCN. LRCN is a model in which a CNN as a feature extractor is applied to each frame before feeding them into an LSTM. In the second framework, we leverage domain knowledge of sign languages to extract the key characteristics of each frame. We first extract 2D landmarks of the hands and then encode them in terms of handshake, hand movement, and hand location. These information are then fed to a Multi-Cue network. We evaluate our proposed models on 52 vocabularies ASLLVD. Each word is played by six signers, and each signer plays each word once. We perform an excessive search on model hyper-parameters such as the number of feature maps, input size, batch size, sequence length, LSTM memory cell, regularization, and dropout. The implementation of the models is publicly accessible through Akandeh (2021).

### 6.1 Introduction

Words in sign languages are mostly expressed by movements of the hands. Some words do not incorporate motion; however, here, we merely refer to dynamic word-level SLR. In the previous chapter, character-level SLR has been studied. In character-level SLR, inputs are in the form of images. In this chapter, we study Dynamic Sign Language Recognition (DSLRL), in which inputs are in the form of videos. Videos have an additional temporal dimension, they are much larger in size, and they may contain different numbers of frames. Although SLR can be viewed as a video classification task, it is much more challenging. SLR includes a large number of classes. Also, in classifying signs, more attention is required in hand regions than the general video classification task. Moreover, one word can be signed differently in different dialects. For example, in ASL, there are over 100 different ways to sign “Birthday” and “Picnic”. There exist many text-to-sign features that translate English text into ASL signs. However, we lack a technology that allows us

to look up corresponding written forms of a given sign.

### **6.1.1 Characteristics**

ASL is a complete and complex language that mainly employs signs made by moving the hands Hoza (2007). Each sign is characterized by manual elements such as shape, movement, and location of the hands. Many ASL signs involve very subtle differences in their three characteristics. For example, the main difference between two signs “small” and “big” is hand movement. Sign “small” involves hands moving toward each other, and sign “big” involves hands moving away from each other. Changing the location of a sign can also completely change the meaning of a sign even though the other parameters are the same. For example, the only parameter change between “summer”, “ugly”, and “dry” is the hand location. A competent SLR framework should be able to take into account those subtle differences and accurately learn the distinguishable information.

### **6.1.2 Challenges**

Word-level SLR shares challenges associated with character-level SLR. Environmental factors such as lighting sensitivity, complex background, occlusion, signee body postures, and camera position are issues that need to be considered at word-level as well. Moreover, in word-level SLR, there is input length variation caused by signing speed. Also, there are a large number of vocabularies that need to be learned. Furthermore, in word-level SLR, publicly available datasets are limited both in quantity and quality despite character-level SLR.

### **6.1.3 Related Work**

To investigate word-level ASL, Huang et al. (2015) proposed an eight layers 3D convolutional neural network. The first four layers were convolution layers followed by sub-sampling and then followed by two fully-connected layers. Color information, depth clue, and body joint positions were fed into the network. They generated their own dataset of 25 vocabularies using Kinect. Each word is played by nine signers, and each signer played each word three times. They stacked

image frames to form a cube for the five types of features (R, G, B, depth, and skeleton) to feed the network. Fifty kernels of the size of  $7 \times 7 \times 5$  were adopted in the convolution layers. They were able to achieve 94% classification result.

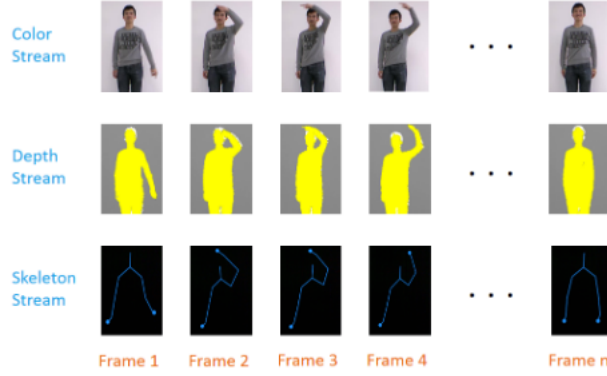


Figure 6.1: Color, depth and skeleton images Huang et al. (2015)

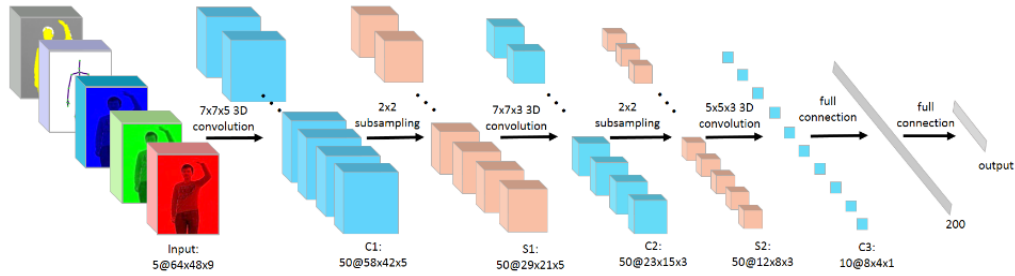


Figure 6.2: The word-level sign recognizer architecture proposed by Huang et al. (2015)

Liu et al. (2016) proposed a seven layers LSTM-based sign language recognition system. They discarded color image and depth information and fed 3D coordinates of hands and elbows into the network. The Kinect sensor camera provides joints skeleton trajectories. The input is a 12-dimensional feature vector comprised of four 3D coordinates of two hands and two elbows. The next layer is an LSTM layer with 512 dimensions. Two fully connected layers with dimensions 512 and 100 come afterward. They evaluated their model on 100 isolated Chinese sign and 500 sign words. The first dataset was performed by five signers who each gestured five times, resulting

in 25,000 images. The second dataset was performed by 50 singers who each gestured five times, resulting in 125,000 images. They were able to achieve 64% and 86%, respectively.

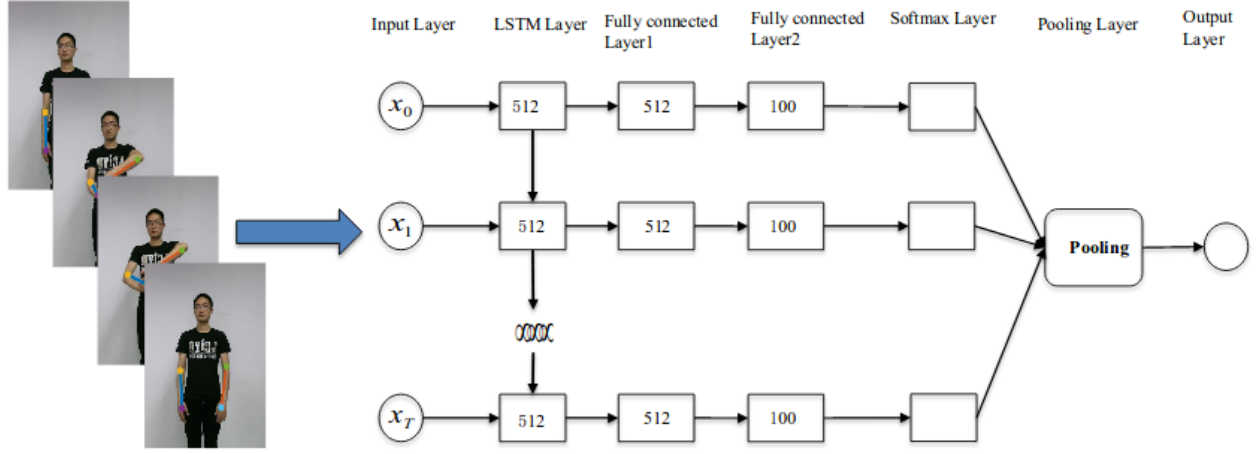


Figure 6.3: The word-level sign recognizer architecture proposed by Liu et al. (2016)

Kumar et al. (2017) proposed a hybrid Hidden Markov Model and Bidirectional Long Short Term Memory on 7500 Indian Sign Language (ISL) gestures comprised of 50 different sign-words. The input was obtained using Leap Motion and the Kinect sensor. They fed horizontal and vertical movement of fingers and palm alongside depth images and were able to achieve 96.2% accuracy.

## 6.2 Proposed Models

In this section, we discuss approaches we took to recognize dynamic isolated signs and propose two deep word-level sign language recognition frameworks. In the first framework, it is aimed to find a robust deep learning algorithm that requires minimum prior knowledge. In the second framework, domain knowledge of sign languages is leveraged to extract the key characteristics of each frame. The first model employs LRCN as its building block and the second model is a multi-cue-based network. We call the first proposed model RSign (R corresponds to raw input) and the second model Multi-Cue Sign (MCSign).



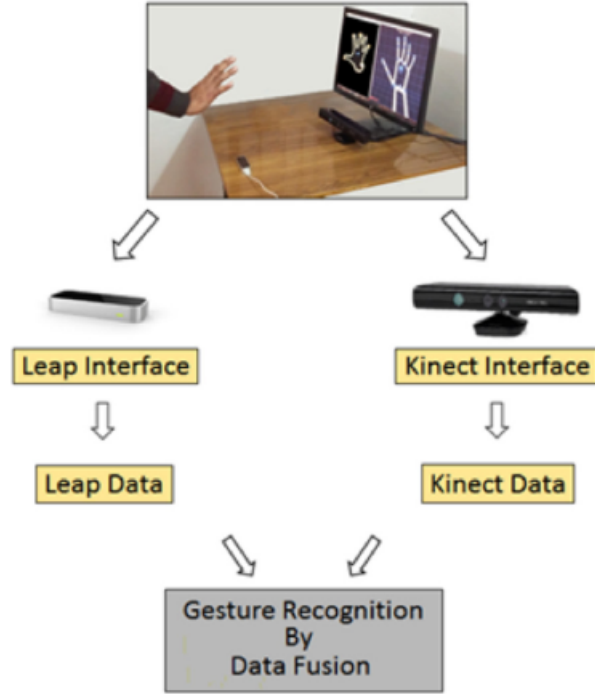


Figure 6.4: The word-level sign recognizer architecture proposed by Kumar et al. (2017)

### 6.2.1 Model Overview

RSign is the sign language application of the LRCN network proposed by Donahue et al. (2017). LRCN is an end-to-end trainable network in which convolutional layers and long-range temporal recursion are combined (figure 2.8).

MCSign, on the other hand, aims to capture and model the three main characteristics associated with each sign. In the first layer, a temporal sequence of 2D landmarks of hands and 2D landmarks of upper-body are captured during the signing. In the second layer, the critical characteristics of the signs, including shape, movement, and location of hands, are modeled. A new 2D black and white image (see figure 6.7) is created out of 2D coordinates of the skeleton joints of hands to model the hands' shape. Hands movement is encoded by spatial displacement of the palm center between two consecutive frames. A one-hot vector corresponding to the relative location of hands to the head is also created to encode the hands' location. In the third level, newly created spatio-temporal trajectories are fed into a CONV-LSTM model. The one-hot vector corresponding to the hand

locations is also fed into an LSTM model. Finally, at the top layer, MCSign adopts a softmax classifier layer. Figure 6.5 provides an overview of the proposed architecture.

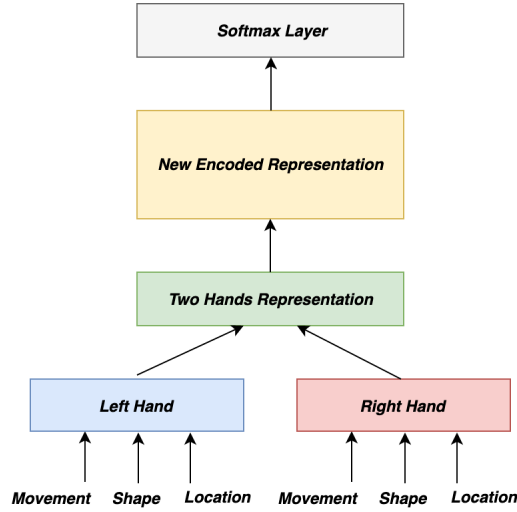


Figure 6.5: MCSign Model Overview

## 6.2.2 Model Details

In this section, a detailed description of two proposed models, namely, RSign and MCSign is provided.

### 6.2.2.1 RSign

The high level architecture of LRCN proposed by Donahue et al. (2017) (figure 2.8) has been used as the building block of RSign model. The more detailed architecture used in this study is given in figure 6.6. Our architecture consists of a default block (blue box) and four repetitions of non-default blocks (red box). We used the Time-Distributed layer in Keras to apply feature extraction (default and non-default blocks) to every temporal slice of the input. Also, we benefited from the LSTM3 model to speed up the training process.

We used the RMSprop optimizer, a batch size of 32, and the categorical cross-entropy loss function. The network specification and also a list of parameters that have been used are given in

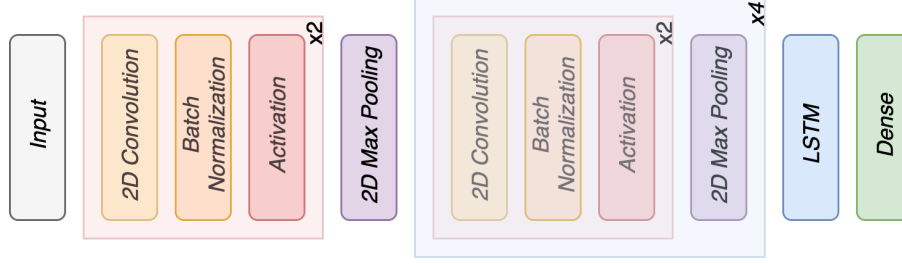


Figure 6.6: RSign model overview

table 6.1. It is observed that even a small sequence length is sufficient to perform well due to the continuous nature of sign language. It is also observed that smaller batch sizes perform better.

Table 6.1: RSign Network specifications.

|  |                           |
|--|---------------------------|
| Input dimension                              | $200 \times 200 \times 3$ |
| 1 <sup>st</sup> layer of default block       | Conv2D(32, 7×7)-Relu      |
| 2 <sup>nd</sup> layer of default block       | Conv2D(32, 3×3)-Relu      |
| Number of feature maps in non-default blocks | 128-64-128-64             |
| LSTM memroy cell                             | 256                       |
| Sequence length                              | None                      |
| Dropout                                      | 0.6                       |
| Regularization                               | 0.001                     |

### 6.2.2.2 MCSign

To emphasize on crucial characteristics of the signs and inject domain-specific expert knowledge into the system, we propose to extract and model each characteristic as three input modalities (cues) to the network. Multi-hands tracking and Pose module in Mediapipe has been used to extract this representative information (see figure 4.1 for the 2D landmarks of hands tracked by Mediapipe).

Having access to 21 2D landmarks of the hands, we create new black and white images representing handshape in each frame. Figure 6.7 illustrates how the ASL signs of the word “Christmas” is precisely captured by the temporal sequence of skeleton joints of the fingers. We also extract hand movement information of both hands as the spatial displacement of the palm center between two consecutive frames. Hand location is also encoded in a one-hot vector based on hand closeness to eyes, mouth, or chest. The extracted characteristics result in six trajectories that

capture handshape, movement, and location. Figure 6.8 illustrates the architecture of the proposed model. The “None” in this figure refers to the number of frames, which can be variable and depends on the signee speed or length of the word.



Figure 6.7: The skeleton joints of ASL sign “Christmas”

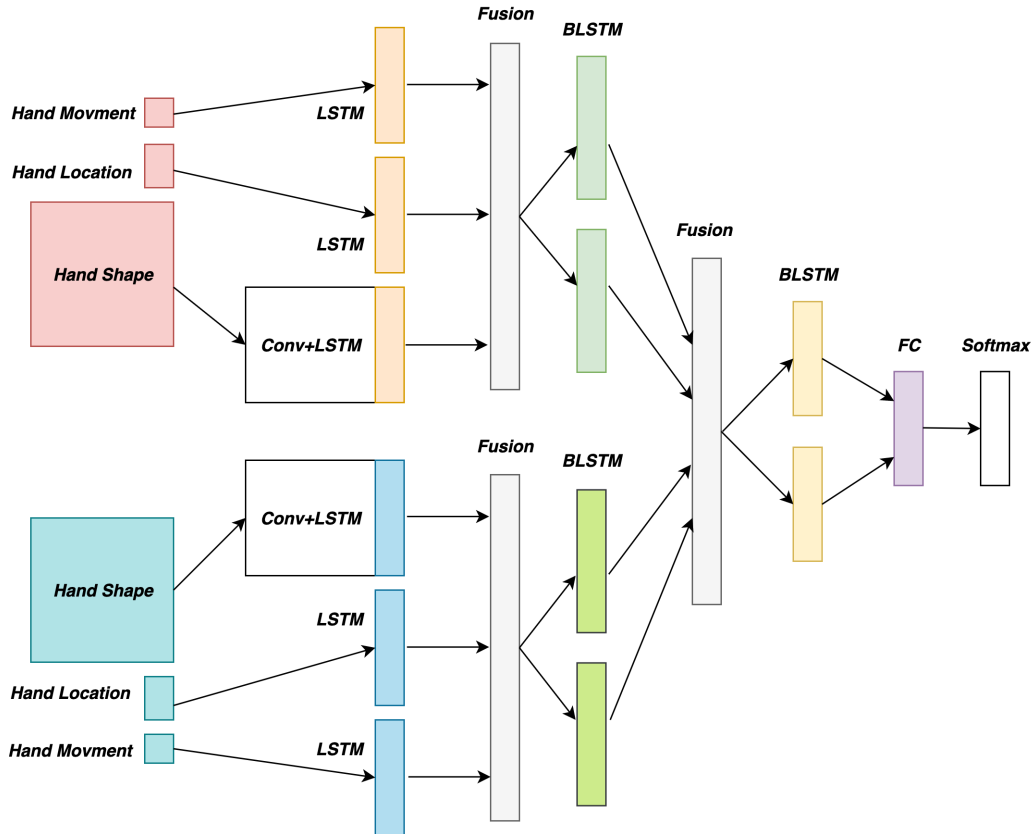


Figure 6.8: MCSign Model Architecture

As shown in figure 6.8, our proposed model consists of seven main layers. Within the first layer,

a block of CONV-LSTM is embedded. This CONV-LSTM model is also LRCN-based in which a CNN model as a feature extractor is applied to each frame before feeding them into an LSTM. For the CNN part, we employ the SConv model that has been proposed in the character-level chapter. In the first layer, the new skeleton images of each hand are fed to the CONV-LSTM block. Hand movement and hand location are also fed into two separate LSTMs. The new representation of each characteristic is then concatenated in the first fusion layer and fed into BLSTM to obtain an integrated representation of each hand. Two high-level hand results are then fused and fed into another BLSTM. Finally, The results of this process are fed into a fully connected layer with a *softmax* nonlinearity that drives the final classification decision. See figure 6.9, for a detailed model summary.

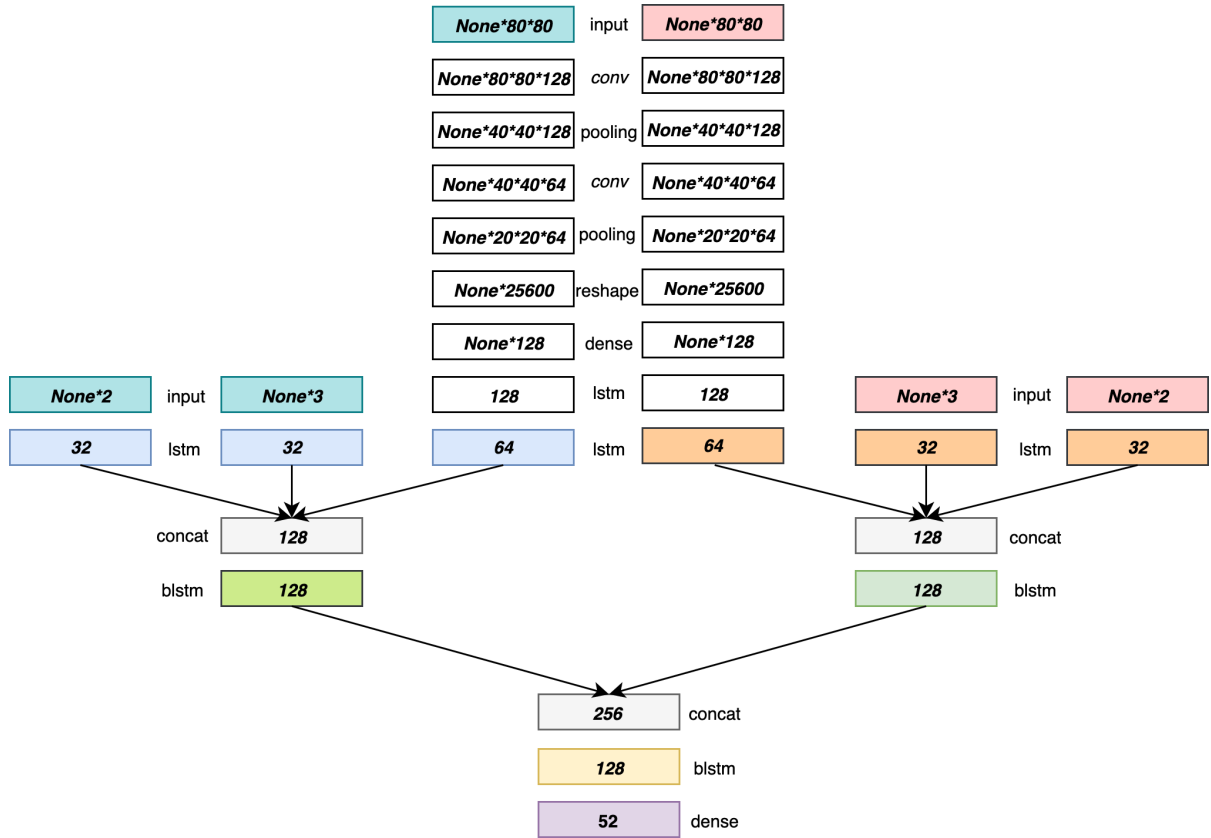


Figure 6.9: MCSign model summary

### 6.2.3 Design Choice

To decide on the building block of both proposed frameworks, we started by employing architecture that has been used by other video classification tasks, namely LRCN, Features+LSTM, C3D, and CONVLSTM. Based on our experiments, LRCN submitted the best performance among all.

Also, to decide on the input form of the first proposed framework, we fed raw frames, optical flow field images (figure 6.10), and also frames with emphasis on hands (6.11) using skin detection techniques. We believe that optical flow field images are too vague for the network. Since signees have worn clothes with different skin exposure levels, the network with input in the form of frames with emphasis on hands failed in giving proper results.

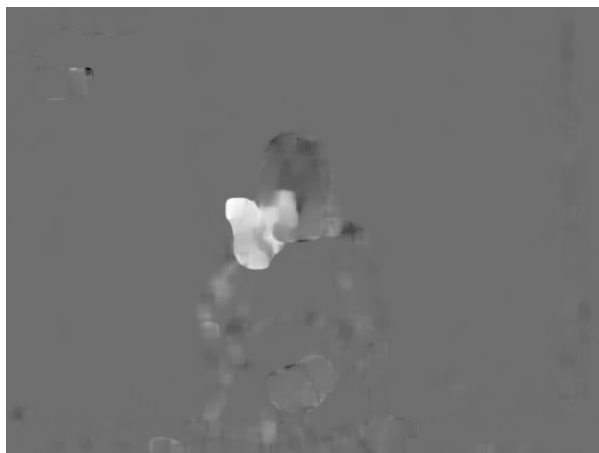


Figure 6.10: Optical flow field sample

Despite character-level SLR, in word-level SLR, publicly available datasets are limited both in quantity and quality. Deep networks require large amounts of labeled training data to demonstrate satisfactory performance. A model with limited parameters cannot learn the problem, and a model with too many parameters can learn it too well. Both scenarios lead to a model that does not generalize well. The complexity of the architecture is a controlling factor in how well a model generalizes. If a model is under-fitted, in which training accuracy is not high enough, one could increase the complexity of the model by enlarging the architecture. If it is overfitted, in which training accuracy is high enough, but testing accuracy is not, one may decrease the complexity. Recently, to reduce generalization error, researchers use a large model along with regularization to

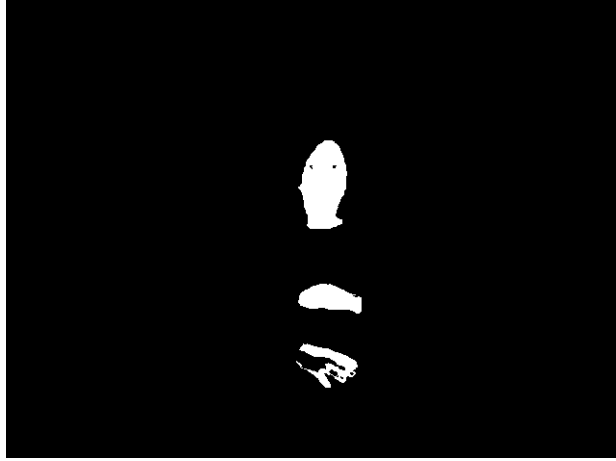


Figure 6.11: Frames with emphasis on hands

put a constraint on weights and avoid overfitting. It has been shown that this approach not only reduces overfitting but also leads to faster optimization.

### 6.3 Dataset

The American Sign Language Lexicon Video Dataset (ASLLVD) Neidle et al. (2012) has been collected at Boston University. It is captured by four synchronized cameras, providing a side view, a close-up, a half-speed front view, and a full-resolution front view (figure 6.12). It consists of more than 3300 ASL signs. Six native signees perform each sign once for a total of almost 9,800 tokens. Gloss labels and sign start and end times are also included in linguistic annotations. In this study, we perform classification on 52 dynamic signs.

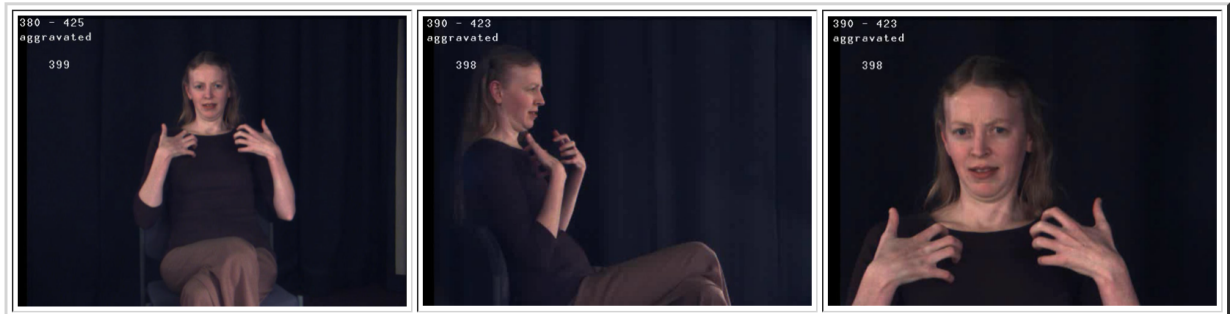


Figure 6.12: ASLLVD Neidle et al. (2012)

## 6.4 Experimental Setup

After downloading and placing videos, FFmpeg was used to extract frames from videos. FFmpeg is an open-source project consisting of a vast software suite of libraries and programs for handling video, audio, and other multimedia files and streams. We also used OpenCV to obtain optical flow images. OpenCV is a library of programming functions mainly aimed at computer vision tasks. The skin detection algorithm is also implemented in OpenCV. Also, since the number of samples per class was limited, we enlarged the dataset size by means of augmentation to a total of 5 per sample.

To implement the CNN block of the CONV-LSTM network, we employed the SConv model that has been proposed in the character-level chapter with TimeDistributed layer in Keras. The only difference was to perform batch-normalization and then apply *relu* nonlinearity after each convolutional layer.

As mentioned before, the MediaPipe palm detection model is invoked only when the landmark model can no longer identify hand presence. To increase the performance, we had to turn this feature off.

There are approximately 40 common handshapes used in ASL, including the 24 static letters of the alphabet 6.13. Before training the proposed model on ASLLVD, we first created a dataset of those handshapes, enlarged the quantity through data augmentation, and then trained the shape-network part of the proposed model on these static hand shapes. Also, to train the shape part of the network on 40 common hand shapes, we initialized it with character-level model (introduced in the previous chapter) weights.

## 6.5 Experimental Results

We used leave-one-subject-out cross-validation to examine the generalization capability of the models across different subjects by selecting five signees for training and left one for evaluation. Later we decided to change the protocol and test it across participants on unseen samples. Overall, MCSing and RSign achieved an average accuracy of 91% and 84% on translating 52 signs across 6





Figure 6.13: 40 common hand shapes used in ASL

participants. This indicates that our proposed models are capable of capturing the key characteristics of the signs. Figures 6.14 and 6.15 show the training and validation accuracy across 6 participants for both proposed models.

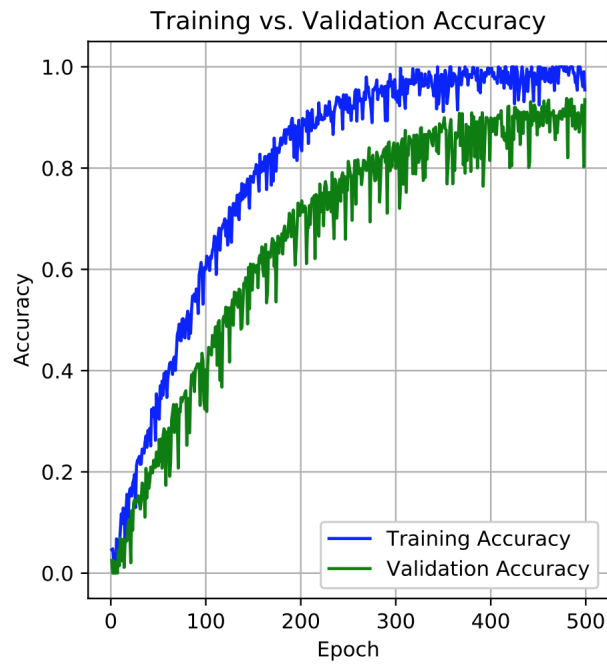


Figure 6.14: Training & Validation Accuracy for the MCSign model on ASLLV dataset.

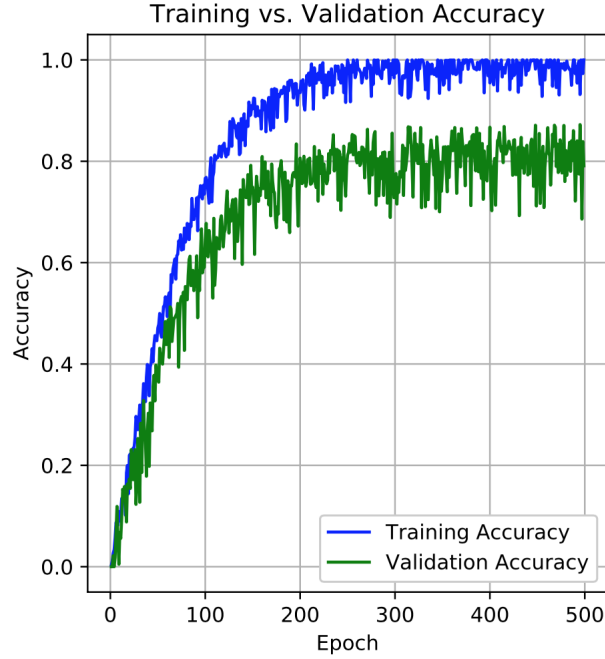


Figure 6.15: Training & Validation Accuracy for the RSign model on ASLLV dataset.

## 6.6 Discussion

With our presented end-to-end model, we are able to classify 52 sign words with 91% accuracy. By creating black and white skeleton images using Mediapipe from raw data, we not only overcome lighting conditions, complex background, and camera position challenges but also create a way to easily combine data from variant sources and hence combat limited data resources in the SLR field. Since the proposed network is insensitive to dataset type, combining different datasets boosts the performance of the network tremendously.

## 6.7 Summary

In this chapter, we proposed two solutions to word-level SLR. In the first approach, no prior knowledge has been leveraged. Raw frames are fed into an 18-layer LRCN. LRCN is a model in which a CNN as a feature extractor is applied to each frame before feeding them into an LSTM. In the second approach, three main characteristics (hand shape, hand position, and hand movement information) associated with each sign have been extracted using Mediapipe. 2D landmarks of

handshape have been used to create the skeleton of the hands and then are fed to a CONV-LSTM block. Hand locations and hand positions as relative distance to head are fed to separate LSTMs. All three sources of information have been then integrated into a Multi-Cue Network. We evaluated the performance of proposed models on 52 vocabularies ASLLVD. Performing an excessive search on model hyper-parameters such as the number of feature maps, input size, batch size, sequence length, LSTM memory cell, regularization, and dropout, we were able to obtain promising results in this domain.

## CHAPTER 7

### SENTENCE-LEVEL SLR

This chapter presents two deep continuous sign language recognition frameworks that map videos of sign language sentences to sequences of gloss labels. Connectionist Temporal Classification (CTC) has been used as the classifier level of both models. CTC is used to avoid pre-segmenting the sentences into individual words. The first model is an LRCN-based model, and the second model is a Multi-Cue network. LRCN is a model in which a CNN as a feature extractor is applied to each frame before feeding them into an LSTM. In the first framework, to minimize prior knowledge, we feed raw frames into an 18-layer LRCN with a CTC on top. In the second framework, we leverage domain knowledge of sign languages to extract the key characteristics of each frame. The features are then fed into a Multi-Cue model with a CTC classification layer. We evaluate proposed models on RWTH-PHOENIX-Weather multi-signer. This dataset contains 5672 sentences in German sign language for training with 65,227 signs and 799,006 frames in total. We perform an excessive search on model hyper-parameters such as the number of feature maps, input size, batch size, sequence length, LSTM memory cell, regularization, and dropout. The implementation of the models is publicly accessible through Akandeh (2021).

#### 7.1 Introduction

In previous chapters, character-level and word-level sign language recognition have been studied. In character-level SLR, inputs are in the form of images, and in word-level SLR, inputs are in the form of videos. In this chapter, we study continuous sign language recognition (CSLR) in which temporal boundaries of the words in the sentences are not defined. Most existing sign language recognition systems can only recognize a single sign at a time and thus requires users to pause between signs, which is not very practical in daily-life interaction. To address this problem, the correspondence between video sequence and sign gloss sequence needs to be learned. Glossing corresponds to mapping signs word-for-word to another written language. Glosses differ from

translation as they only denote the meaning of each part in a sign language sentence and do not necessarily construct a grammatically correct sentence in the written language. In this chapter, we do not address linguistic structures and grammar unique to sign language.

### **7.1.1 Characteristics**

As mentioned in the previous chapter, each sign is characterized by manual elements such as shape, movement, and location of the hands. To structure sentences, non-manual elements like eye gaze, mouth shape, facial expression, and body pose are also involved.

Facial expressions are used to prevent confusion or misunderstandings. The visible mouth shapes can add information to the meaning of a sign and make it distinguishable from a semantically related sign. Some ASL signs have a permanent mouth morpheme as part of their production. For example, the ASL word NOT-YET requires a mouth morpheme (TH), whereas LATE has no mouth morpheme. These two are the same signs but with a different non-manual signal.

In American Sign Language, eye gazing serves a variety of functions. It can regulate turn-taking and mark constituent boundaries. Eye gazing is also frequently used to repair or monitor utterances and to direct the addressee's attention. It is also engaged in indexing and in expressing object and subject agreement and definiteness versus indefiniteness (Thompson (2006)).

In this chapter, non-manual elements, despite their importance, have not been addressed. In the next chapter, we have made some suggestions and introduced some references on how to involve those essential factors in recognition systems.

### **7.1.2 Challenges**

As mentioned in previous chapters, there are many challenges associated with the SLR task. Publicly available datasets are limited both in quantity and quality. Environmental factors such as lighting sensitivity, complex background, occlusion, signee body postures, and camera position are also challenging issues. In terms of sign linguistics, there are subtle differences between different signs, and there are a large number of vocabularies that need to be learned. Specifically,

in sentence-level SLR, there is no word alignment, and temporal boundaries of a specific word are not clear. Also, there is sentence length variation caused by the number of the word or by signing speed. Moreover, signs are context-dependent and, coarticulation in which sign is affected by the preceding or following signs also plays an important role. All these factors together make sentence-level SLR a very challenging task.

### 7.1.3 Related Work

Koller et al. (2016a) employed a pre-trained 22-layer CNN model within an iterative EM algorithm on a sequence of data. The algorithm iteratively refined the frame-level annotation and subsequent training of the CNN. Three thousand manually labeled hand shape images of 60 different classes were employed to train the model.

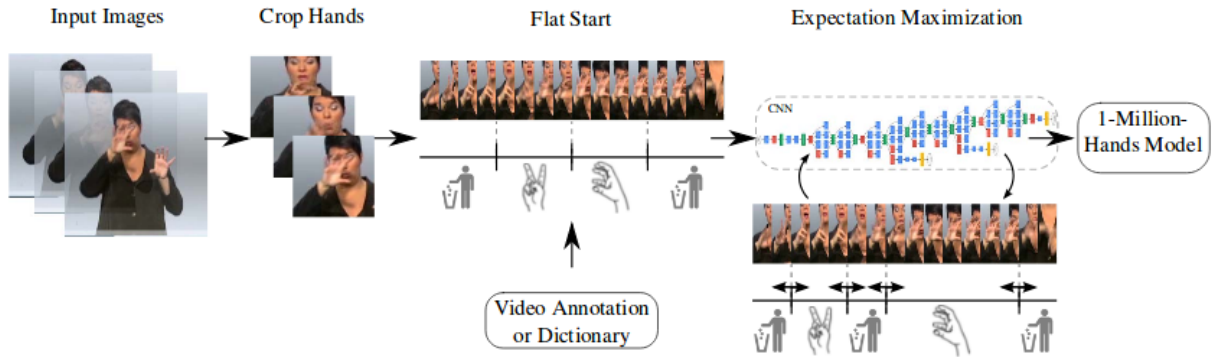


Figure 7.1: The sentence-level sign recognizer architecture proposed by Koller et al. (2016a)

Koller et al. (2016b) embedded a CNN into an HMM. The outputs of the CNN were treated as Bayesian posteriors, and they trained the system in an end-to-end fashion. Their model was very similar to the LRCN mentioned earlier, which combined CNN and LSTM. They were able to improve the state-of-the-art accuracy on three challenging continuous sign language benchmarks (SIGNUM , RWTH-PHOENIX-Weather2012 , RWTH-PHOENIX-Weather Multi-signer ) by 15%, 38% and 13.3% respectively.

Cui et al. (2017) used a CNN-LSTM model to learn the mapping of sign sequences to sequences of the gestures on RWTH-PHOENIX-Weather Multi-signer and achieved 61.3% accuracy. Their

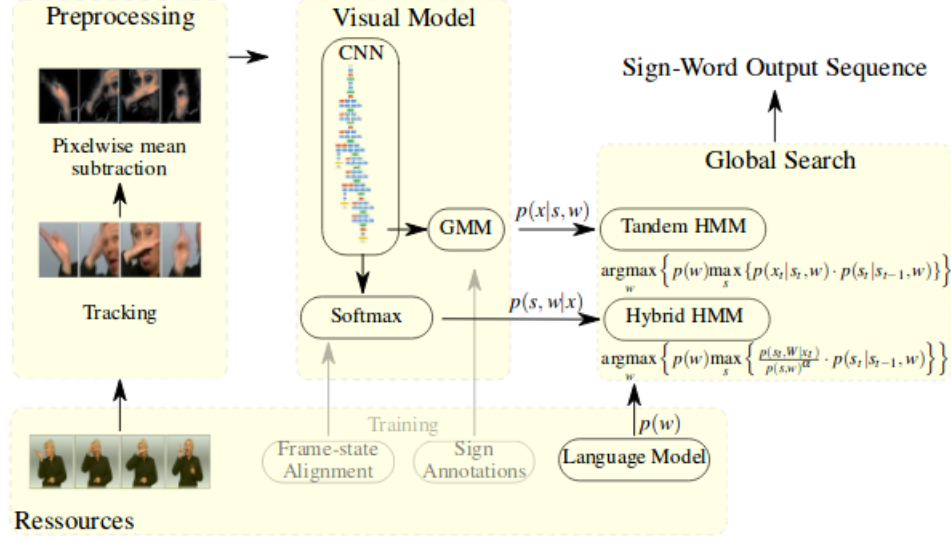


Figure 7.2: The sentence-level sign recognizer architecture proposed by Koller et al. (2016b)

architecture consists of a CNN with temporal convolution and spatial pooling, a bidirectional LSTM for global sequence learning, and a detection network. The detection network combines the temporal convolution operations on the spatio-temporal features. According to the authors, this combination acts like a sliding window along with the sign sequences.

To perform Sign2Gloss task (SLR), Camgoz et al. (2020) utilized a pre-trained Inception model as spatial embeddings in a CNN+LSTM+HMM setup. They extracted frame level representations from sign videos and trained two-layered sign language transformers to learn CSLR (continuous sign language recognition) and SLT(sign language translation) jointly in an end-to-end manner. Niu & Mak (2020) also proposed to use the transformer encoder as the contextual model for CSLR to fine-tune the lower-level visual feature extractor during model training. To improve model robustness, they proposed dropping video frames stochastically (SFD) and randomly stopping the gradients of some frames during training (SGS).

## 7.2 Proposed Models

We present the development and implementation of two deep sentence-level sign language recognition frameworks. In the first framework, to minimize prior knowledge, raw frames are fed

to a deep network. In the second framework, sign language linguistics is leveraged to extract the key characteristics of each frame. In both models, we adopt a probabilistic framework based on Connectionist Temporal Classification to overcome pre-segmentation requirements posed by the sequence learning problem associated with sentence-level SLR. We call the first proposed model RSign-C (R corresponds to raw input and C corresponds to CTC), and the second model Multi-Cue Sign-C (MCSign-C).

### 7.2.1 Model Overview

In this section, we provide an overview of two proposed models, namely, RSign-C and MCSign-C. These models are based on models that have been proposed in the previous chapter for word-level SLR, except we modify the *softmax* layer, which calculates the probability of a single sign and adds a CTC layer that calculates the probabilities of a sequence of signs. To have a standalone chapter, we re-present the architecture of both models.

RSign-C is an 18-layers LRCN with CTC layer on top. In the LRCN network, a CNN model as a feature extractor is applied to each frame before feeding them into an LSTM (2.8).

MCSign-C, on the other hand, aims to capture and model the three main characteristics associated with each sign. In the first layer, a temporal sequence of 2D landmarks of hands, as well as 2D landmarks of the upper-body, are captured during the signing. In the second layer, the critical characteristics of the signs, including shape, movement, and location of hands, are modeled. A new 2D black and white image (see figure 7.3) is created out of 2D coordinates of the skeleton joints of hands to model the handshapes. Hands movement is encoded by spatial displacement of the palm center between two consecutive frames. A one-hot vector corresponding to the relative location of hands to the head is also created to encode the hands' location. In the third level, newly created spatio-temporal trajectories are fed into a CONV-LSTM model. The one-hot vector corresponding to the hand locations is also fed into an LSTM model. Finally, at the top layer, MCSign-C adopts a CTC-based approach to output sequences of gloss labels. Figure 7.4 provides an overview of the proposed architecture.



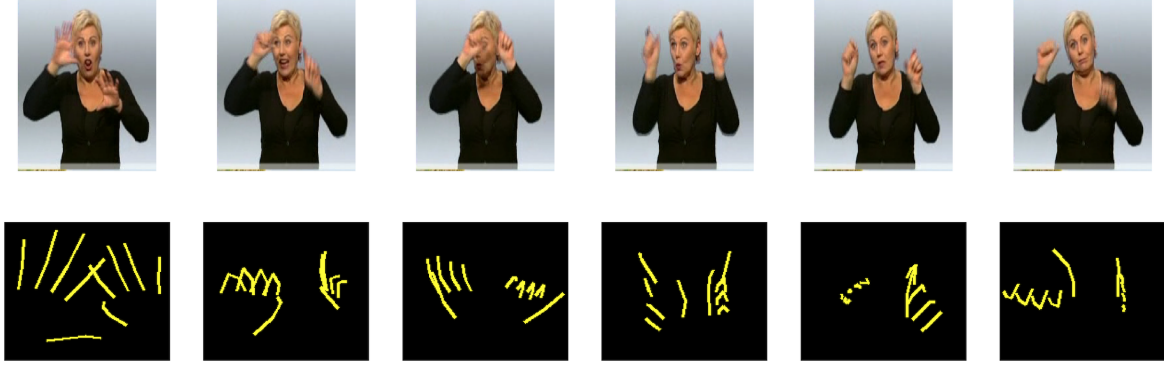


Figure 7.3: A 2D black and white image sequence created from PHOENIX dataset

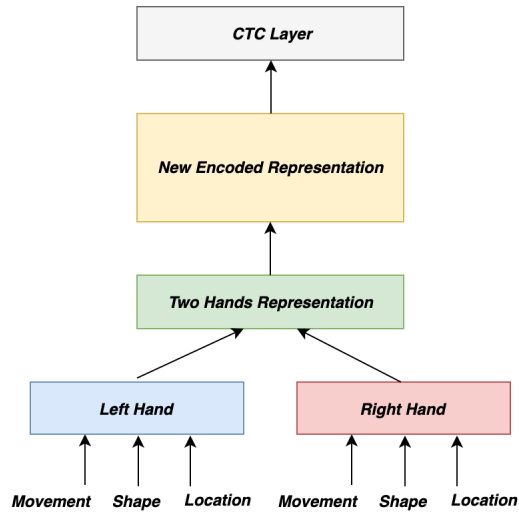


Figure 7.4: MCSign-C model overview

## 7.2.2 Model Details

This section provides a detailed description of two proposed models, namely, RSign-C and MCSign-C.

### 7.2.2.1 RSign-C

We modify the RSign model that has been proposed in the previous chapter such that it is capable of computing the probabilities of a sequence of signs. LRCN proposed by Donahue et al. (2017) (figure 2.8) has been used as the building block of RSign model. To produce a probability distribution over all labels at each time step, we modify the final LSTM to return all sequence corresponds

to each hidden state, then apply a *softmax* function and finally add a CTC layer. The more detailed architecture used in this study is given in figure 7.5. As mentioned earlier, we used the Time-Distributed layer in Keras to apply feature extraction to every temporal slice of the input.

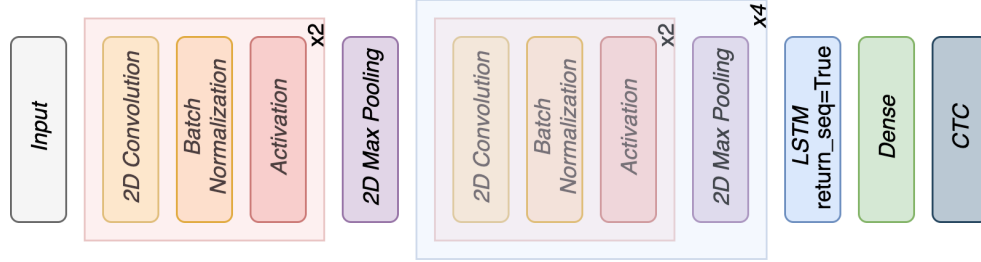


Figure 7.5: RSign-C model overview

### 7.2.2.2 MCSign-C

To emphasize on crucial characteristics of the signs and inject domain-specific expert knowledge into the system, we propose to extract and model each characteristic as three input modalities (cues) to the network. Multi-hands tracking and pose modules in Mediapipe have been used to extract this representative information. (see figure 4.1 for the 2D landmarks of hands tracked by Mediapipe).

Having access to 21 2D landmarks of the hands, we create new black and white images representing handshape in each frame. We also extract hand movement information of both hands as the spatial displacement of the palm center between two consecutive frames. Hand location is also encoded in a one-hot vector based on hand closeness to eyes, mouth, or chest. The extracted characteristics result in six trajectories that capture information related to each handshape, movement, and location. Figure 7.6 illustrates the architecture of the proposed model.

As shown in figure 7.6, our proposed model consists of seven main layers. Within the first layer, a block of CONV-LSTM is embedded. This CONV-LSTM model is also LRCN-based in which a CNN model as a feature extractor is applied to each frame before feeding them into an LSTM. For the CNN part, we employ the SConv model that has been proposed in the character-level chapter. In the first layer, the new skeleton images of each hand are fed to the CONV-LSTM block. Hand movement and hand location are also fed into two separate LSTM. The new representation

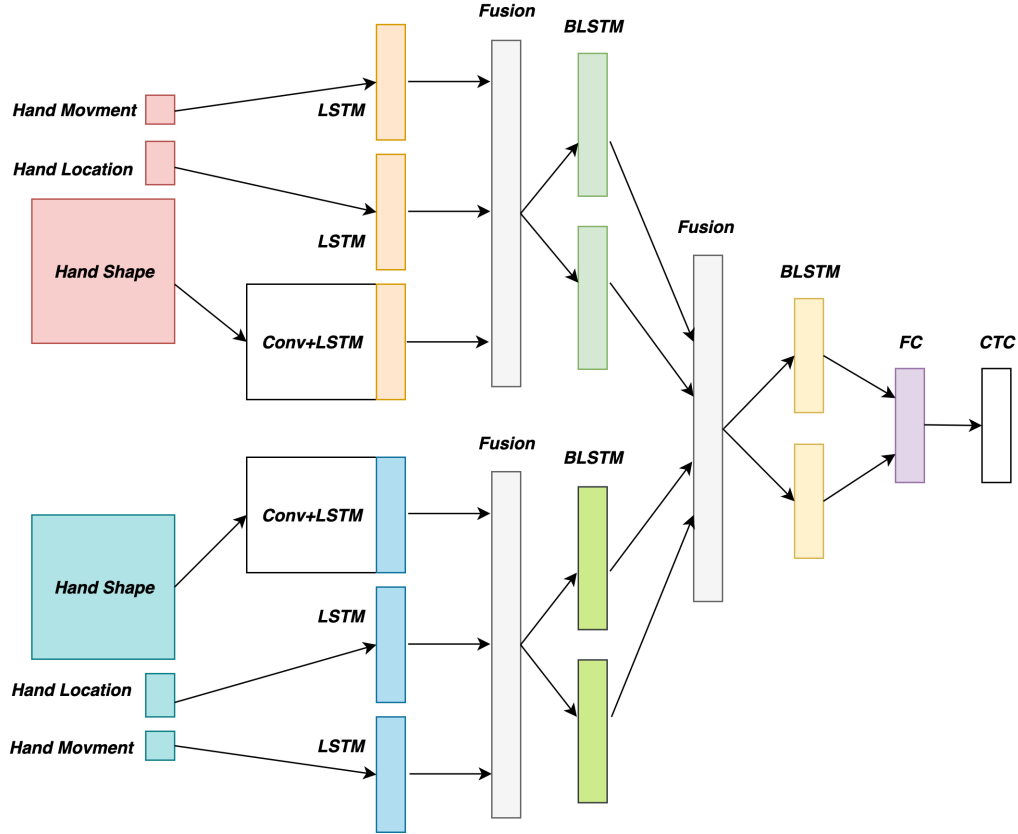


Figure 7.6: MCSign-C architecture

of each characteristic is then concatenated in the first fusion layer and fed into BLSTM to obtain an integrated representation of each hand. Two high-level hand results are then fused and fed into another BLSTM. Finally, The results of this process are fed into a fully connected layer with CTC loss that drives the final classification decision. See figure 7.7, for a detailed model summary. The “None” in this figure refers to the number of frames, which can be variable and depends on the signee speed or length of the sentence.

In the CTC approach, the probabilities of all the possible sentences formed by the word included in the output domain are computed. This approach not only eliminates the need for word pre-segmentation and post-processing but also addresses variable-length sequences Fang et al. (2017). To obtain the final gloss label, as it was discussed in chapter 2, section 4, adjacent duplicates and all the blank symbols in the inferred label sequence are removed.

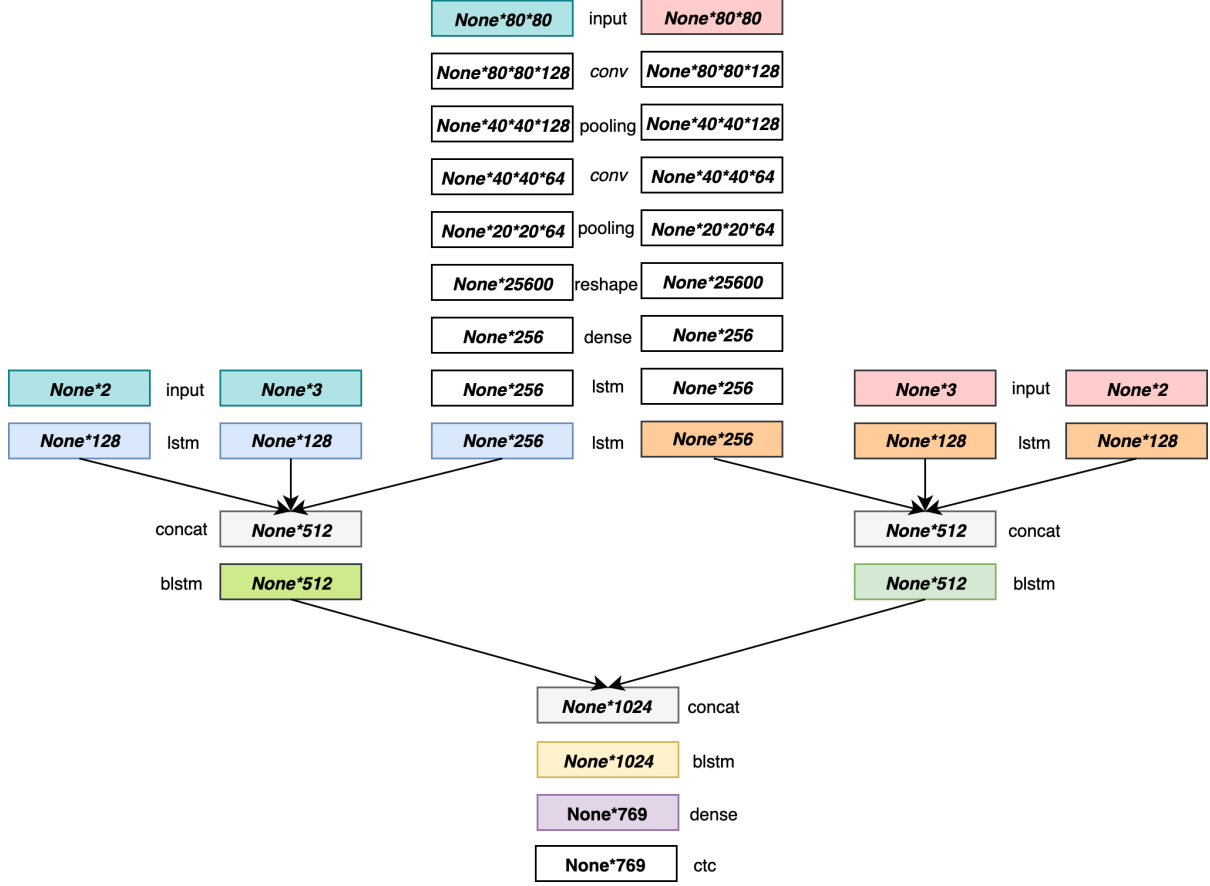


Figure 7.7: MCSign-C model summary

### 7.2.3 Design Choice

There have been many different architectures proposed by many researchers to address the SLR task. Here, we present the main reasons behind selecting some specific design choices.

In the MCSign-C model, we leverage sign language linguistics to extract trajectories of representative information from the frame sequences during the signing and develop models based on the extracted information. Handshape information of both left and right hands is encoded by zero-centering the palm center of the hands and then scaling the given normalized 2D coordinates of the skeleton joints. Since there was no ultimate solution to layout the coordinates, we decided to leave it to the network to decide and proposed to regenerate each frame using 2D landmarks. This method also handles different hand size issues imposed by distance to the camera.

As many signs share similar characteristics at the beginning of their trajectories, to avoid any

confusion by traditional unidirectional LSTM, a bidirectional LSTM model has been adopted (also suggested by Fang et al. (2017)). A bidirectional LSTM model performs inference based on both past and future information and computes the hidden state sequence by combining the output sequences of LSTM by iterating forwards and backward.

To perform sentence-level SLR, current technologies employ a framework that requires pre-segmenting individual words within the sentence. This restriction requires users to pause between adjacent signs when signing sentences. To address this problem, some researchers have proposed sign boundary detection to detect when a sign ends, and the next one begins. Attention mechanisms (Encoder-Decoder Networks) and Connectionist Temporal Classification (CTC) have been also proposed by other researchers to compute the probability of the whole sentence directly. Since CTC is the key technique that drives the modern automatic speech recognition systems such as Amazon Alexa and Apple Siri Fang et al. (2017), we propose to adopt a framework based on CTC which removes the requirement of pre-segmentation and can be easily built on top of the MCSign.

### 7.3 Dataset

The RWTH-PHOENIX-Weather multi-signer Forster et al. (2012), is a dataset for continuous sign language recognition. This dataset contains 5672 sentences in German sign language for training with 65,227 signs and 799,006 frames in total. Nine signers perform these videos. Table 7.1 summarizes setup statistics RWTH-PHOENIX-Weather (2012).

Table 7.1: RWTH-PHOENIX Setup Statistics

|             |                            | Glosses | German |
|-------------|----------------------------|---------|--------|
| Train       | number of sentences        | 2612    | 2612   |
|             | number of running words    | 20713   | 26585  |
|             | vocabulary size            | 768     | 1389   |
|             | singeltons/vocabulary size | 32.4%   | 36.4%  |
| Development | number of sentences        | 250     | 250    |
|             | number of running words    | 2573    | 3293   |
|             | out-of-vocabulary-words    | 1.4%    | 1.9%   |
| Test        | number of sentences        | 228     | 228    |
|             | number of running words    | 2163    | 2980   |
|             | out-of-vocabulary-words    | 1.0%    | 1.5%   |

## 7.4 Experimental Setup

To implement the CNN block of the CONV-LSTM network, we employed the SConv model that has been proposed in the character-level chapter with TimeDistributed layer in Keras. The only difference was to perform batch-normalization and then apply *relu* nonlinearity after each convolutional layer.

As mentioned before, in the CTC layer, a probability distribution over all labels at each time step is predicted. This can be implemented by feeding a 2D input (e.g., the output of an LSTM layer where return-sequence is true) to a Softmax function and then employ the CTC loss function.

To infer a likely output, the Beam Search algorithm has been used. Only a predetermined number of best partial solutions (beam size) are kept as candidates in the beam search algorithm. To get the optimum beam size performing error analysis are required. If the probability of a true label  $y^*$  for a given input is lower than the probability of generated output  $\tilde{y}$ , then the network is at fault, and more training is required. If  $p(y^*) > p(\tilde{y})$ , then the beam search is at fault, and the beam size needs to be increased.

## 7.5 Experimental Results

To evaluate the performance of the two proposed models, we use word error rate (WER) as the evaluation metric. WER is a standard metric of the performance of a sentence-level translation or a speech recognition system and is computed as the minimum number of word insertions, substitutions, and deletions required to get from the reference to the hypothesis, divided by the number of words in the reference. To further investigate Slim LSTM models proposed in chapter 3, we then run multiple experiments with MCSign-C-Slim models and compare them against baseline MCSign-C.

### 7.5.1 MCSign-C & RSign-C

MCSign-C achieves an average 35.2% word error rate on mapping Dev sentences and an average of 35.3% word error rate on Test sentences. Table 7.2 lists other approaches, and WER reported

on RWTH-PHOENIX-Weather dataset.

Table 7.2: Comparison between various approaches on the RWTH-PHOENIX dataset

| Method                              | Dev(WER) | Test(WER) |
|-------------------------------------|----------|-----------|
| CMLLR Koller et al. (2015)          | 55.0     | 53.0      |
| 1-Mio-Hands Koller et al. (2016b)   | 45.1     | 47.1      |
| CNN-Hybrid Koller et al. (2016b)    | 38.3     | 38.8      |
| SubUNets Camgoz et al. (2017)       | 40.8     | 40.7      |
| Staged-Opt Cui et al. (2017)        | 39.4     | 38.7      |
| Re-sign Koller et al. (2017)        | 27.1     | 26.8      |
| LS-HAN Huang et al. (2018)          | –        | 38.3      |
| Dilated Pu et al. (2018)            | 38.0     | 37.3      |
| Hybrid CNN-HMM Koller et al. (2018) | 31.6     | 32.5      |
| IAN Pu et al. (2019)                | 37.1     | 36.7      |
| DenseTCN Guo et al. (2019)          | 35.9     | 36.5      |
| DNF Cui et al. (2019)               | 23.8     | 24.4      |
| DNF Cui et al. (2019)               | 23.1     | 22.9      |
| CNN-LSTM-HMM Koller et al. (2020)   | 26.0     | 26.0      |
| RSign                               | 45.1     | 45.1      |
| MCSign                              | 35.2     | 35.3      |

### 7.5.2 MCSign-C-Slim

Ten Slim LSTM models have been introduced and investigated in chapter 3. Due to their promising results, we decide to evaluate and verify them further on RWTH-PHOENIX-Weather dataset. We train the MCSign-C-Slim model with three configurations: LSTM1, LSTM3, and LSTM6. We then compare baseline performance to the proposed models. To train each model, we perform parameter selection for learning rate, number of epochs, and beam width.

For all three models, we only replaced LSTM units with corresponding Slim models. The networks were trained by RMSprop optimizer with a batch size of 32. We first investigate the effect of the beam width. The beam width varied from 1 to 6, then set to the optimum value of 4 when only the 4 most probable hypotheses are stored. Similarly, the number of epochs for the training was changed from 100 to 1000.

We then investigate how LSTM1, LSTM3 and LSTM6 affect WER score. Figure 7.8 shows that Slim models and the baseline model have nearly the same word error rates. This indicates that

the proposed models, although being lightweight, do not harm the model performance.

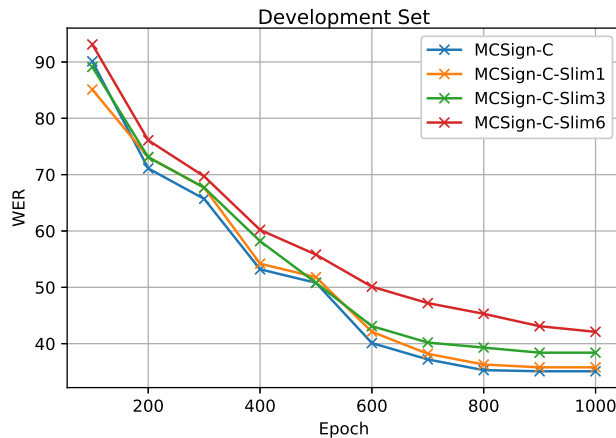


Figure 7.8: MCSign-C-Slim models comparison

In Table 7.3, we list the time per epochs for each model. As the table indicates, slim models provide significant improvements over the baseline model in terms of training cost. The model is implemented with the Tensorflow.keras deep-learning platform and trained on a Nvidia K80 with 4.1 TFLOPS as the GPU resource (Google Colab Setting).

Table 7.3: Time per epoch for each MCSign-C-Slim model.

| Model          | Time per Epoch (s) |
|----------------|--------------------|
| MCSign-C-Slim1 | 36                 |
| MCSign-C-Slim3 | 30                 |
| MCSign-C-Slim6 | 29                 |
| MCSign-C       | 40                 |

## 7.6 Discussion

With our presented end-to-end model, we are able to achieve 35 WER on WTH-PHOENIX-Weather dataset. The slightly lower performance of the model is compensated by greater robustness to environmental circumstances, like lighting. By creating black and white skeleton images using Mediapipe from raw data, we not only overcome lighting conditions, complex background, and camera position challenges but also create a way to easily combine data from variant sources and



hence combat limited data resources in the SLR field. Since the proposed network is insensitive to dataset type, combining different datasets boosts the network’s performance tremendously. Experimental results indicate that our model achieves substantial improvements over mainstream methods in terms of training speed.

## **7.7 Summary**

In this chapter, we proposed two solutions to sentence-level SLR. Sentence-level SLR required mapping videos of sign language sentences to sequences of gloss labels. Connectionist Temporal Classification (CTC) has been used as the classifier level of both models. CTC is used to avoid pre-segmenting the sentences into individual words. The first model is an LRCN-based model, and the second model is a Multi-Cue Network. LRCN is a model in which a CNN as a feature extractor is applied to each frame before feeding them into an LSTM. In the first approach, no prior knowledge has been leveraged. Raw frames are fed into an 18-layer LRCN with a CTC on top. In the second approach, three main characteristics (hand shape, hand position, and hand movement information) associated with each sign have been extracted using Mediapipe. 2D landmarks of handshape have been used to create the skeleton of the hands and then are fed to a CONV-LSTM model. Hand locations and hand positions as relative distance to head are fed to separate LSTMs. All three sources of information have been then integrated into a Multi-Cue network with a CTC classification layer. We evaluated the performance of proposed models on RWTH-PHOENIX-Weather and were able to achieve 35 WER on RWTH-PHOENIX-Weather dataset.

## CHAPTER 8

### FUTURE ROADMAP

Although this study aimed to address many aspects of SLR that computer scientists have overlooked, there are still significant gaps to be addressed. Specifically, non-manual elements, including the eye gaze, mouth shape, facial expression, require much attention in SLR. Face Mesh and Iris tracking modules in MediaPipe can be beneficial to pursue this track.

To extend the SLR task, one may also look into SLT. SLT differs from SLR as the latter merely detects a sequence of signs without taking into account the linguistic structures and grammar unique to sign language Yin (2020). There have been limited studies on mapping glosses to spoken language. To map the detected glosses into a proper sentence in the target language Neural Machine Translation (NMT) has been introduced. An encoder-decoder architecture, also known as a sequence to sequence model, is primarily employed in recent NMT approaches. However, sequence to sequence networks are unable to model long-term dependencies in large input sentences. To address this issue, attention mechanisms are introduced Bahdanau et al. (2015). Transformer Vaswani et al. (2017) is an encoder-decoder network in which self-attention layers are used in place of recurrent networks. The next phase in translating sign language to written or spoken language can be piping the output of an SLR system into a sequence to sequence model. RWTH-PHOENIX-Weather is currently the only publicly available dataset with both gloss labels and spoken language translations.

## **BIBLIOGRAPHY**

## BIBLIOGRAPHY

- Adithya, V & R Rajesh. 2020. A deep convolutional neural network approach for static hand gesture recognition. *Procedia Computer Science* 171. 2353–2361.
- Akandeh, Atra. 2019. Slim lstm models. <https://github.com/atrakriv/Slim-LSTMs> .
- Akandeh, Atra. 2021. Sign language recognition. <https://github.com/atrakriv/SLR> .
- Aly, Saleh, Basma Osman, Walaa Aly & Mahmoud Saber. 2016. Arabic sign language fingerspelling recognition from depth and intensity images. In *2016 12th international computer engineering conference (icenco)*, vol. 2016, 104.
- Bahdanau, Dzmitry, Kyunghyun Cho & Yoshua Bengio. 2015. Neural machine translation by jointly learning to align and translate. In *Iclr 2015 : International conference on learning representations 2015*, .
- Bengio, Y., A. Courville & P. Vincent. 2013. Representation learning: A review and new perspectives. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8). 1798–1828.
- Bengio, Yoshua. 2009. Learning deep architectures for ai .
- Bengio, Yoshua, Pascal Lamblin, Dan Popovici & Hugo Larochelle. 2006. Greedy layer-wise training of deep networks. In *Advances in neural information processing systems 19*, 153–160.
- Boulanger-Lewandowski, Nicolas, Yoshua Bengio & Pascal Vincent. 2012. Modeling temporal dependencies in high-dimensional sequences: Application to polyphonic music generation and transcription. *arXiv preprint arXiv:1206.6392* .
- Bragg, Danielle, Oscar Koller, Mary Bellard, Larwan Berke, Patrick Boudreault, Annelies Braf-fort, Naomi Caselli, Matt Huenerfauth, Hernisa Kacorri, Tessa Verhoef, Christian Vogler & Meredith Ringel Morris. 2019. Sign language recognition, generation, and translation: An interdisciplinary perspective. In *The 21st international acm sigaccess conference on computers and accessibility*, 16–31.
- Camgoz, Necati Cihan, Simon Hadfield, Oscar Koller & Richard Bowden. 2017. Subunets: End-to-end hand shape and continuous sign language recognition. In *2017 ieee international conference on computer vision (iccv)*, 3075–3084.
- Camgoz, Necati Cihan, Oscar Koller, Simon Hadfield & Richard Bowden. 2020. Sign language transformers: Joint end-to-end sign language recognition and translation. In *2020 ieee/cvf conference on computer vision and pattern recognition (cvpr)*, 10023–10033.
- Chung, Junyoung, Caglar Gulcehre, KyungHyun Cho & Yoshua Bengio. 2014. Empirical evaluation of gated recurrent neural networks on sequence modeling. *arXiv preprint arXiv:1412.3555* .

- Cui, Runpeng, Hu Liu & Changshui Zhang. 2017. Recurrent convolutional neural networks for continuous sign language recognition by staged optimization. In *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 1610–1618.
- Cui, Runpeng, Hu Liu & Changshui Zhang. 2019. A deep neural framework for continuous sign language recognition by iterative training. *IEEE Transactions on Multimedia* 21(7). 1880–1891.
- Deng, J., W. Dong, R. Socher, L.-J. Li, K. Li & L. Fei-Fei. 2009. ImageNet: A Large-Scale Hierarchical Image Database. In *Cvpr09*, .
- Donahue, Jeff, Lisa Anne Hendricks, Marcus Rohrbach, Subhashini Venugopalan, Sergio Guadarrama, Kate Saenko & Trevor Darrell. 2017. Long-Term Recurrent Convolutional Networks for Visual Recognition and Description. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 39(4). 677–691.
- Fang, Biyi, Jillian Co & Mi Zhang. 2017. Deepasl: Enabling ubiquitous and non-intrusive word and sentence-level sign language translation. In *Proceedings of the 15th ACM conference on embedded network sensor systems*, 5.
- Forster, Jens, Christoph Schmidt, Thomas Hoyoux, Oscar Koller, Uwe Zelle, Justus Piater & Hermann Ney. 2012. Rwth-phoenix-weather: A large vocabulary sign language recognition and translation corpus. In *Proceedings of the eighth international conference on language resources and evaluation (lrec-2012)*, 3785–3789.
- Gaidon, Adrien, Zaid Harchaoui & Cordelia Schmid. 2013. Temporal localization of actions with actoms. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(11). 2782–2795.
- Goodfellow, Ian, Yoshua Bengio & Aaron Courville. 2016. Deep learning. <http://www.deeplearningbook.org> .
- Graves, Alex, Santiago Fernandez, Faustino Gomez & Jurgen Schmidhuber. 2006. Connectionist temporal classification: labelling unsegmented sequence data with recurrent neural networks. In *Proceedings of the 23rd international conference on machine learning*, 369–376.
- Greff, Klaus, Rupesh K. Srivastava, Jan Koutnik, Bas R. Steunebrink & Jurgen Schmidhuber. 2017. Lstm: A search space odyssey. *IEEE transactions on Neural Networks and Learning Systems* 28(10). 2222–2232.
- Guo, Dan, Shuo Wang, Qi Tian & Meng Wang. 2019. Dense temporal convolution network for sign language translation. In *Proceedings of the twenty-eighth international joint conference on artificial intelligence*, 744–750.
- He, Kaiming, Xiangyu Zhang, Shaoqing Ren & Jian Sun. 2015. Deep residual learning for image recognition. *arXiv preprint arXiv:1512.03385* .
- Hinton, Geoffrey E., Simon Osindero & Yee-Whye Teh. 2006. A fast learning algorithm for deep belief nets. *Neural Computation* 18(7). 1527–1554.

- Hinton, Geoffrey E., Nitish Srivastava, Alex Krizhevsky, Ilya Sutskever & Ruslan Salakhutdinov. 2012. Improving neural networks by preventing co-adaptation of feature detectors. *arXiv preprint arXiv:1207.0580*.
- Hoza, Jack. 2007. *It's not what you sign, it's how you sign it: Politeness in american sign language*.
- Huang, Jie, Wengang Zhou, Houqiang Li & Weiping Li. 2015. Sign language recognition using real-sense. In *2015 ieee china summit and international conference on signal and information processing (chinasip)*, 166–170.
- Huang, Jie, Wengang Zhou, Qilin Zhang, Houqiang Li & Weiping Li. 2018. Video-based sign language recognition without temporal segmentation. In *Aaai*, 2257–2264.
- Ioffe, Sergey & Christian Szegedy. 2015. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *Proceedings of the 32nd international conference on machine learning*, 448–456.
- Jaworek-Korjakowska, Joanna, Pawel Kleczek & Marek Gorgon. 2019. Melanoma thickness prediction based on convolutional neural network with vgg-19 model transfer learning. In *2019 ieee/cvf conference on computer vision and pattern recognition workshops (cvprw)*, 0–0.
- Jia, Yangqing, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross B. Girshick, Sergio Guadarrama & Trevor Darrell. 2014. Caffe: Convolutional architecture for fast feature embedding. In *Proceedings of the 22nd acm international conference on multimedia*, 675–678.
- Johnson, Melvin, Mike Schuster, Quoc V. Le, Maxim Krikun, Yonghui Wu, Zhifeng Chen, Nikhil Thorat, Fernanda B. Viégas, Martin Wattenberg, Greg Corrado, Macduff Hughes & Jeffrey Dean. 2016. Google's multilingual neural machine translation system: Enabling zero-shot translation. <http://arxiv.org/abs/1611.04558> abs/1611.04558.
- Karpathy, Andrej, George Toderici, Sanketh Shetty, Thomas Leung, Rahul Sukthankar & Li Fei-Fei. 2014. Large-scale video classification with convolutional neural networks. *IEEE Conference on Computer Vision and Pattern Recognition* 1725–1732.
- Koller, Oscar, Necati Cihan Camgoz, Hermann Ney & Richard Bowden. 2020. Weakly supervised learning with multi-stream cnn-lstm-hmms to discover sequential parallelism in sign language videos. *IEEE Transactions on Pattern Analysis and Machine Intelligence* 42(9). 2306–2320.
- Koller, Oscar, Jens Forster & Hermann Ney. 2015. Continuous sign language recognition: Towards large vocabulary statistical recognition systems handling multiple signers. *Computer Vision and Image Understanding* 141. 108–125.
- Koller, Oscar, Hermann Ney & Richard Bowden. 2016a. Deep hand: How to train a cnn on 1 million hand images when your data is continuous and weakly labelled. In *2016 ieee conference on computer vision and pattern recognition (cvpr)*, 3793–3802.
- Koller, Oscar, Sepehr Zargaran & Hermann Ney. 2017. Re-sign: Re-aligned end-to-end sequence modelling with deep recurrent cnn-hmms. In *2017 ieee conference on computer vision and pattern recognition (cvpr)*, 3416–3424.

- Koller, Oscar, Sepehr Zargaran, Hermann Ney & Richard Bowden. 2018. Deep sign: Enabling robust statistical continuous sign language recognition via hybrid cnn-hmms. *International Journal of Computer Vision* 126(12). 1311–1325.
- Koller, Oscar Tobias Anatol, Sepehr Zargaran, Hermann Ney & Richard Bowden. 2016b. Deep sign: Hybrid cnn-hmm for continuous sign language recognition. In *British machine vision conference 2016*, .
- Krizhevsky, Alex, Ilya Sutskever & Geoffrey E Hinton. 2012. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems* .
- Kuehne, H., H. Jhuang, E. Garrote, T. Poggio & T. Serre. 2011. Hmdb: A large video database for human motion recognition. In *2011 international conference on computer vision*, 2556–2563.
- Kumar, Pradeep, Himaanshu Gauba, Partha Pratim Roy & Debi Prosad Dogra. 2017. A multimodal framework for sensor based sign language recognition. *Neurocomputing* 259. 21–38.
- Larochelle, Hugo, Yoshua Bengio, Jerome Louradour & Pascal Lamblin. 2009. Exploring strategies for training deep neural networks. *Journal of Machine Learning Research* 10. 1–40.
- LeCun, Yann, Patrick Haffner, Leon Bottou & Yoshua Bengio. 1999. Object recognition with gradient-based learning. *Shape, Contour and Grouping in Computer Vision* 319–345.
- Lee, Honglak, Chaitanya Ekanadham & Andrew Y. Ng. 2007. Sparse deep belief net model for visual area v2. In *Advances in neural information processing systems 20*, 873–880.
- Lee, Tai Sing & David Bryant Mumford. 2003. Hierarchical bayesian inference in the visual cortex. *Journal of The Optical Society of America A-optics Image Science and Vision* 20(7). 1434–1448.
- Li, Shao-Zi, Bin Yu, Wei Wu, Song-Zhi Su & Rong-Rong Ji. 2015. Feature learning based on sae pca network for human gesture recognition in rgbd images. *Neurocomputing* 151. 565–573.
- Liu, Tao, Wengang Zhou & Houqiang Li. 2016. Sign language recognition with long short-term memory. In *2016 ieee international conference on image processing (ictp)*, 2871–2875.
- Lu, Y. & F. Salem. 2017. Simplified gating in long short-term memory (lstm) recurrent neural networks. *arXiv:1701.03441* .
- MediaPipe. 2019. MediaPipe Dev. <https://mediapipe.dev> .
- Neidle, Carol, Ashwin Thangali & Stan Sclaroff. 2012. Challenges in development of the american sign language lexicon video dataset (asllvd) corpus .
- Ng, Joe Yue-Hei, Matthew Hausknecht, Sudheendra Vijayanarasimhan, Oriol Vinyals, Rajat Monga & George Toderici. 2015. Beyond short snippets: Deep networks for video classification. In *2015 ieee conference on computer vision and pattern recognition (cvpr)*, 4694–4702.
- Niu, Zhe & Brian Mak. 2020. Stochastic fine-grained labeling of multi-state sign glosses for continuous sign language recognition. In *European conference on computer vision*, 172–186.

- Oyedotun, Oyebade K. & Adnan Khashman. 2017. Deep learning in vision-based static hand gesture recognition. *Neural Computing and Applications* 28(12). 3941–3951.
- Pu, Junfu, Wengang Zhou & Houqiang Li. 2018. Dilated convolutional network with iterative optimization for continuous sign language recognition. In *Ijcai'18 proceedings of the 27th international joint conference on artificial intelligence*, 885–891.
- Pu, Junfu, Wengang Zhou & Houqiang Li. 2019. Iterative alignment network for continuous sign language recognition. In *2019 IEEE/CVF conference on computer vision and pattern recognition (CVPR)*, 4165–4174.
- Pugeault, Nicolas & Richard Bowden. 2011. Spelling it out: Real-time asl fingerspelling recognition. In *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 1114–1119.
- Ranzato, Marc'aurelio, Christopher Poultney, Sumit Chopra & Yann L. Cun. 2006. Efficient learning of sparse representations with an energy-based model. In *Advances in neural information processing systems 19*, 1137–1144.
- Rastgoo, Razieh, Kourosh Kiani & Sergio Escalera. 2018. Multi-modal deep hand sign language recognition in still images using restricted boltzmann machine. *Entropy* 20(11). 809.
- Rioux-Maldague, Lucas & Philippe Giguere. 2014. Sign language fingerspelling classification from depth and color images using a deep belief network. In *2014 Canadian Conference on Computer and Robot Vision*, 92–97.
- Rosenblatt, F. 1958. The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review* 65(6). 386–408.
- RWTH-PHOENIX-Weather. 2012. The RWTH-PHOENIX-Weather Database of German Sign Language. <https://www-i6.informatik.rwth-aachen.de/forster/database-rwth-phoenix.php>.
- Salem, Fathi M. 2016. A basic recurrent neural network model. *arXiv preprint arXiv:1612.09022*.
- Salem, Fathi M. 2018. Slim lstms. *arXiv: Neural and Evolutionary Computing*.
- Salem, Fathi M. 7.11.2016. Reduced parameterization of gated recurrent neural networks. *MSU Memorandum*.
- Sign Language and Static gesture. 2018. Sign Language and Static-Gesture Recognition using scikit-learn. <https://github.com/mon95/Sign-Language-and-Static-gesture-recognition-using-sklearn>.
- Sign Language Club. 2012. American Sign Language Alphabet. <https://www.cchsvoice.org/join-sign-language-club/>.
- Sign Language MNIST. 2018. Drop-In Replacement for MNIST for Hand Gesture Recognition Tasks. <https://www.kaggle.com/datamunge/sign-language-mnist>.
- Simonyan, Karen & Andrew Zisserman. 2014. Two-Stream Convolutional Networks for Action Recognition in Videos. *Neural Information Processing Systems* 27. 568–576.



- Simonyan, Karen & Andrew Zisserman. 2015. Very deep convolutional networks for large-scale image recognition. In *Iclr 2015 : International conference on learning representations 2015*, .
- Soomro, Khurram, Amir Roshan Zamir & Mubarak Shah. 2012. Ucf101: A dataset of 101 human actions classes from videos in the wild. *arXiv preprint arXiv:1212.0402* .
- Szegedy, Christian, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke & Andrew Rabinovich. 2014. Going deeper with convolutions. *arXiv preprint arXiv:1409.4842* .
- Thompson, Robin L. 2006. *Eye gaze in american sign language : linguistic functions for verbs and pronoun*: dissertation.
- Tran, Du, Lubomir Bourdev, Rob Fergus, Lorenzo Torresani & Manohar Paluri. 2015. Learning Spatiotemporal Features with 3D Convolutional Networks. *IEEE International Conference on Computer Vision* 4489–4497.
- Vaswani, Ashish, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser & Illia Polosukhin. 2017. Attention is all you need. In *Proceedings of the 31st international conference on neural information processing systems*, vol. 30, 5998–6008.
- Vitruvian Man. 1490. The proportions of the human body according to Vitruvius. [https://en.wikipedia.org/wiki/Vitruvian\\_Man](https://en.wikipedia.org/wiki/Vitruvian_Man) .
- Wang, Limin, Yu Qiao & Xiaoou Tang. 2014. Latent hierarchical model of temporal structure for complex activity classification. *IEEE Transactions on Image Processing* 23(2). 810–822.
- Wang, Limin, Yuanjun Xiong, Zhe Wang, Yu Qiao, Dahua Lin, Xiaoou Tang & Luc van Gool. 2016. Temporal segment networks: Towards good practices for deep action recognition. In *European conference on computer vision*, vol. 9912 LNCS, 20–36.
- Yin, Kayo. 2020. Sign language translation with transformers. *arXiv preprint arXiv:2004.00588* .
- Zaremba, Wojciech. 2015. An empirical exploration of recurrent network architectures. *An empirical exploration of recurrent network architectures* .
- Zeiler, Matthew D. & Rob Fergus. 2014. Visualizing and understanding convolutional networks. In *13th european conference on computer vision, eccv 2014*, 818–833.
- Zhu, Yi, Zhenzhong Lan, Shawn Newsam & Alexander G. Hauptmann. 2017. Hidden two-stream convolutional networks for action recognition. *arXiv preprint arXiv:1704.00389* .