DEEPSIGN: A DEEP-LEARNING ARCHITECTURE FOR SIGN LANGUAGE RECOGNITION

by

JAI SHAH

Presented to the Faculty of the Graduate School of

The University of Texas at Arlington in Partial Fulfillment

of the Requirements

for the Degree of

MS in Computer Science

THE UNIVERSITY OF TEXAS AT ARLINGTON

December 2018

Acknowledgements

I would like to convey my warmest gratitude to my supervising professor Dr. Vassilis Athitsos for giving me the opportunity to conduct research on this project and for his constant guidance, support, encouragement.

I want to thank my committee members Dr. Farhad Kamangar and Dr. Christopher Conley for their interest in my research and for taking the time to serve in my dissertation committee.

Special thanks to Ph.D. student Srujana Gattupalli for constantly supporting me and letting me use her machine for performing experiments.

I would also like to extend my appreciation to the CSE department for providing me with all the facilities and infrastructure necessary to carry out my master's studies.

I would like to thank my beloved parents, who have taught me the value of hard work and education. I would like to thank all my friends for their affection and care that brings color to my life. Finally, I would like to thank all whose direct and indirect support has helped me in completing my thesis.

September 13, 2018

Abstract

DEEPSIGN: A DEEP-LEARNING ARCHITECTURE FOR SIGN LANGUAGE RECOGNITION

Jai Shah, MS

The University of Texas at Arlington, 2018

Supervising Professor: Vassilis Athitsos

Sign languages are used by deaf people for communication. In sign languages, humans use hand gestures, body, facial expressions and movements to convey meaning. Humans can easily learn and understand sign languages, but automatic sign language recognition for machines is a challenging task. Using recent advances in the field of deep-learning, we introduce a fully automated deep-learning architecture for isolated sign language recognition. Our architecture tries to address three problems: 1) Satisfactory accuracy with limited data samples 2) Reducing chances of over-fitting when the data is limited 3) Automating recognition of isolated signs. Our architecture uses deep convolutional encoder-decoder architecture for capturing spatial information and LSTM architecture for capturing temporal information. With a vocabulary of 14 one-handed signs chosen from LSA64 Dataset, our architecture achieves an accuracy of 96.02% for top 3 predictions in signer dependent settings and an accuracy of 77.85% for top 3 predictions in signer independent settings.

Table of Contents

List of Illustrations

List of Tables

Chapter 1

Introduction

Sign Language Recognition is a difficult task for Computers. For humans, Sign Language Recognition is an easy task. Humans look for the location of the hands and the motion of the hands relative to each other for recognizing the Sign. More generally, Sign Language Recognition is a spatio-temporal learning problem which humans solve by locating where hands are at current timestep and remember the location of the hands in the previous timesteps by which humans can figure out the motion of the hands and recognize the sign.

In this paper, we present DeepSign, a deep learning architecture for spatio-temporal learning to recognize discrete Sign Language. Contrary to other deep learning approaches for Sign Language Recognition which requires lots of training data to learn, our method is specifically designed to learn from limited training data.

The rest of the paper is as follows: In Chapter 2 we go over related work on Sign Language Recognition. In Chapter 3 we formulate the problem and give an overview of our method. In Chapter 4 we go over the Dataset. In Chapter 5 we briefly describe our deep learning architecture in detail. Then we go over the results that our model achieved in Chapter 6. In Chapter 7 we compare the performance of our model with traditional Dynamic Time Warping method on Sign Language Recognition. Finally, we conclude the paper in Chapter 7 by discussing our model and possible future work.

Chapter 2

Related Work

Sign Language Recognition is not a new problem in the field of Computer Vision. In the 90's researchers used [1] Hidden Markov Approach for Sign Language Recognition because of their capability to capture temporal information. Researchers have also used extra hardware sensors [2], [3] as feature extractors and use classifiers on the extracted features. Above mentioned work dealt with static data and was not real-time. With the increasing popularity of deep neural networks and their capability to provide high-performance, many researchers started using these new deep-learning techniques [4], [5] for sign language recognition which gave them better performance but still, they were dealing with static data. Recently, [6], [7], [8] have achieved real-time sign recognition performance with complex convolutional neural networks and they have worked on dynamic data. Deep Neural Network methods achieve high accuracy; however, they need a lot of training data. Previous literature has few shortcomings that we address with our work:

- Introduce DeepSign, a new deep-learning architecture which achieves comparable results with limited training data
- Automate sign-language recognition for isolated signs

Chapter 3

Problem Formulation and Overview of Method

Given a query video say *Q* we want to find its corresponding sign *S*. Our final goal is to learn a function *A: Q* →*S* that maps any query video *Q* belonging to a domain *X* to its corresponding sign *S*. Query video *Q* is a sequence of frames = $(q_1, q_2, \dots, q_t)$. This becomes a many-to-one problem because we want to map multiple inputs in a sequence to a single output.

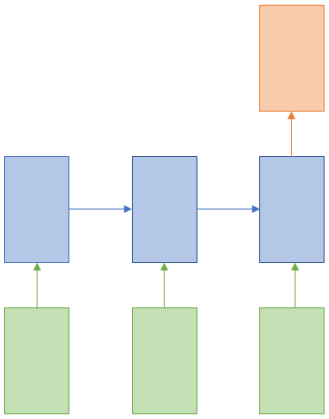


Figure 3-1 Many-To-One problem.

So, we can re-write learning function *A: Q* → *S* to *A:* $(q_1, q_2, \dots, q_t)$ →*S* where we simply substitute *Q* to $(q_1, q_2, \dots, q_t)$. We want a first model, that we call Model-1, to learn a function *E: Q* → *F* that maps input sequence $(q_1, q_2, \dots, q_t)$ to feature sequence $(f_1, f_2, \dots, f_t)$ by minimizing the Mean Squared Error loss function:

$$M.S.E_{Loss} = (x - y)^2 \qquad (1)$$

where ***x*** is the image pixels which is fed as input to the Model-1 and ***y*** is the pixels predicted by the Model-1.

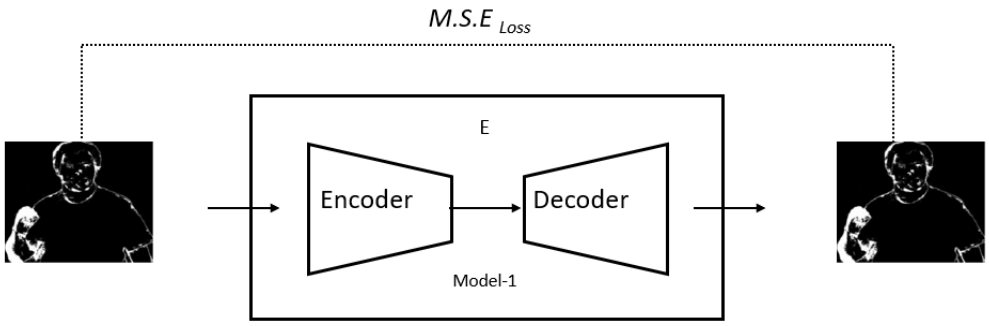


Figure 3-2: Model-1 Overview to learn E: Q → F by minimizing M.S.E Loss.

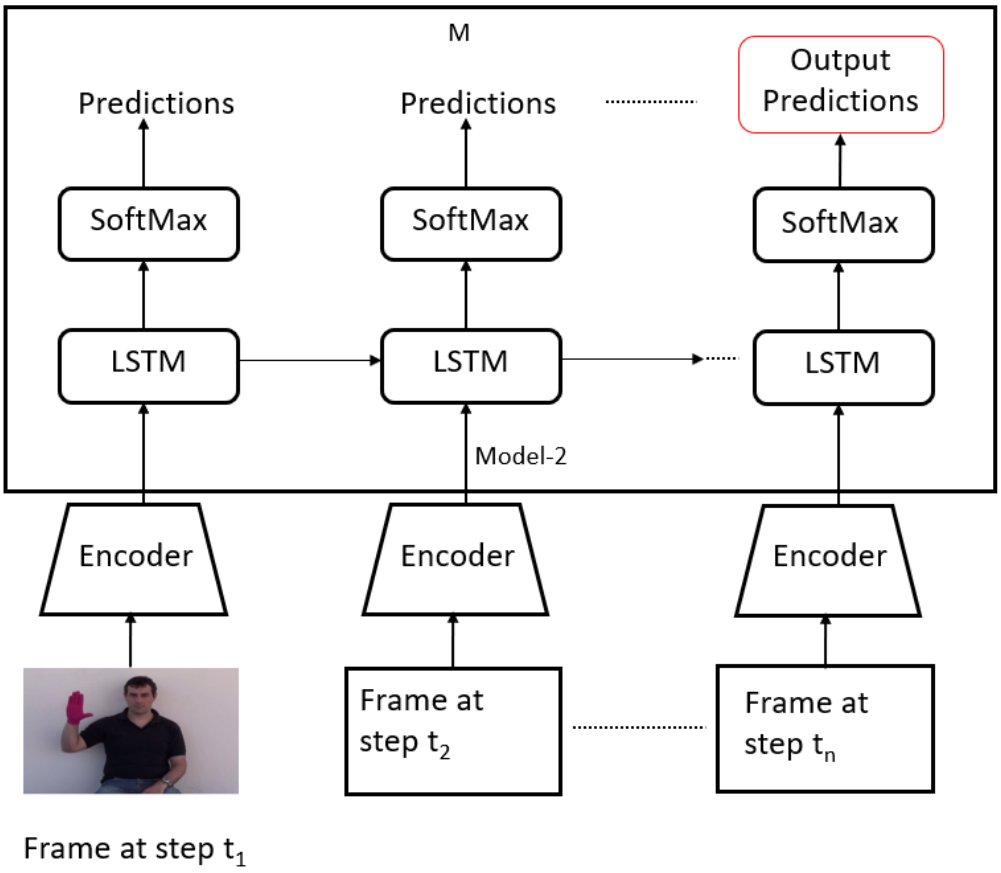


Figure 3-3: Model-2 Overview to learn M: F → S by minimizing Cross-Entropy. To calculate the error, the prediction in the last time step is used.

We want a second model, that we call Model-2, to learn a function *M: F* → *S* that maps feature sequence $(f_1, f_2, \dots, f_t)$ to its sign *S* by minimizing the Cross-Entropy loss function:

$$Cross - Entropy_{Loss} = -\sum p(x) log\, q(x) \qquad (2)$$

where *p(x)* is the probability distribution of actual target and *q(x)* is probability distribution predicted by the Model-2. By combining Model-1 and Model-2 together we wish to learn the goal function *A: Q* → *S.*

Chapter 4

Dataset

We used LSA64 dataset [9] to evaluate our model's performance. Originally the dataset includes 3200 videos. 10 non-expert subjects were asked to execute 5 repetitions of 64 different types of signs.

ID	Name	H	ID	Name	H	ID	Name	H	ID	Name	H
01	Opaque	R	17	Call	R	33	Hungry	R	49	Yogurt	B
02	Red	R	18	Skimmer	R	34	Map	B	50	Accept	B
03	Green	R	19	Bitter	R	35	Coin	B	51	Thanks	B
04	Yellow	R	20	Sweet milk	R	36	Music	B	52	Shut down	R
05	Bright	R	21	Milk	R	37	Ship	R	53	Appear	B
06	Light-blue	R	22	Water	R	38	None	R	54	To land	B
07	Colors	R	23	Food	R	39	Name	R	55	Catch	B
08	Red	R	24	Argentina	R	40	Patience	R	56	Help	B
09	Women	R	25	Uruguay	R	41	Perfume	R	57	Dance	B
10	Enemy	R	26	Country	R	42	Deaf	R	58	Bathe	B
11	Son	R	27	Last name	R	43	Trap	B	59	Buy	R
12	Man	R	28	Where	R	44	Rice	B	60	Copy	B
13	Away	R	29	Mock	B	45	Barbecue	B	61	Run	B
14	Drawer	R	30	Birthday	R	46	Candy	R	62	Realize	R
15	Born	R	31	Breakfast	B	47	Chewing-gum	R	63	Give	B
16	Learn	R	32	Photo	B	48	Spaghetti	B	64	Find	R

Table - 4.1: The H column is the hand column. The column H specifies whether the sign was performed with the right hand or both hands. Only the first 14 signs highlighted by red border were used for the experimentation.

These 64 chosen were the signs which are commonly used in LSA lexicon which includes nouns and verbs. Table - 4.1 shows the list of signs which has id, name and hands as columns for each sign. It consists of 42 one-handed signs and 22 two-handed signs. We have only used first 14 one-handed signs for recognition. Gloves were used by each subject. Each subject wore a magenta colored glove in the right hand and a fluorescent green colored glove in the left hand. Few samples of the dataset can be seen in Figure 4.1.

Figure 4.1: Sample snapshots from LSA64 Dataset.

The 14 signs we used consisted of 700 videos in total. Each sign has 50 videos. For every sign, 20 videos were used for testing and 30 videos were used for training. In total out of 700 videos and we used 280 videos for testing and 430 videos for training. No extra information of gloves was used in our pipeline. Our pipeline can be used for any isolated sign language data.

Chapter 5 Approach

Our Approach has 1) Preprocessing step and 2)Training step.

5.1 Preprocessing step

In the preprocessing step, we used the [10] SSD model pre-trained by TensorFlow API on the COCO dataset to detect the person in each individual frame. We apply background subtraction on each frame to capture the motion information and we extract the pixels within the bounding-box provided by the SSD model. We find the edges in the frame and combine the background subtraction frame with the edge frame. We finally normalize the pixel values to be either 0 or 1.

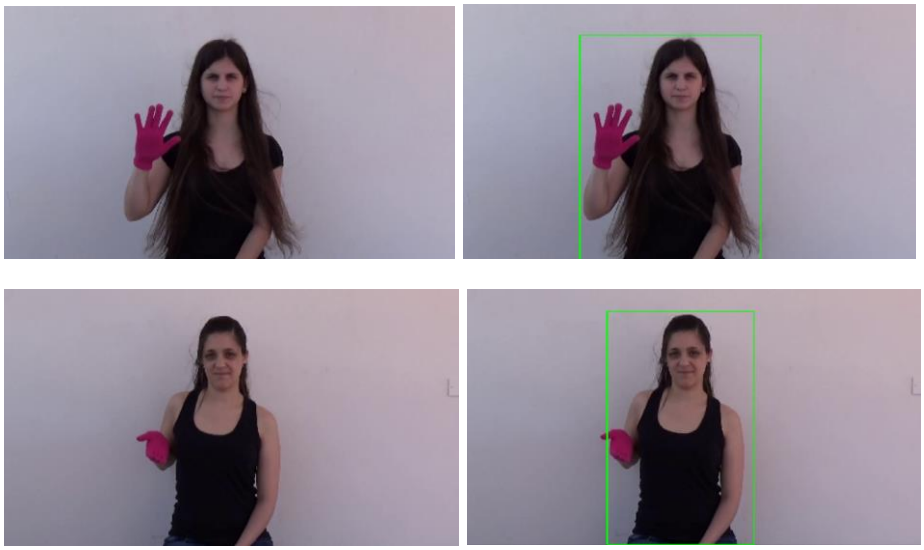


Figure 5-1: Object detection results.

(a)

(b)

Figure 5-2: Preprocessing step results.

5.2 Training Step

The training step of our pipeline is a two-stage training. Training is performed from scratch, no pre-trained information is used. In the first stage of training, we use a Convolutional Encoder-Decoder Architecture to capture the spatial information. In the second stage, we use a LSTM network to capture the temporal information.

5.2.1 Stage-1 Training Architecture

The stage-1 training uses Model-1 for training to learn the function $E: Q \rightarrow F$. Model-1 is an Encoder-Decoder Architecture.

Layer	Kernel	Stride	Padding	In-size	Out-size
Covolution1	3x3	1x1	SAME	240 x 240 x 1	240 x 240 x 16
Leaky Relu				240 x 240 x 16	240 x 240 x 16
Inception-1				240 x 240 x 16	240 x 240 x 16
Max-Pooling-1	2x2	2x2	SAME	240 x 240 x 16	120 x 120 x 16
Inception-2				120 x 120 x 16	120 x 120 x 8
Max-Pooling-2	2x2	2x2	SAME	120 x 120 x 8	60 x 60 x 8
Inception-3				60 x 60 x 8	60 x 60 x 8
Max-Pooling-3	2x2	2x2	SAME	60 x 60 x 8	30 x 30 x 8
Fully-Connected-1				30 x 30 x 8	1x512
Leaky Relu				1x512	1x512

Table 5-1: Structure of the encoder network. The input and output size are described in *rows x cols x # filters*. The structure is inspired by inception[11].

The Encoder-Decoder architecture performs the reconstruction. The Encoder learns the hidden representation of every frame and the Decoder tries to regenerate the frame from the hidden representation. Our Encoder Network's architecture is based on [11] Inception Architecture. The Encoder Network is a series of convolutional layers and max-pooling layers, to down-sample the original input which is at the end followed by a fully connected layer.

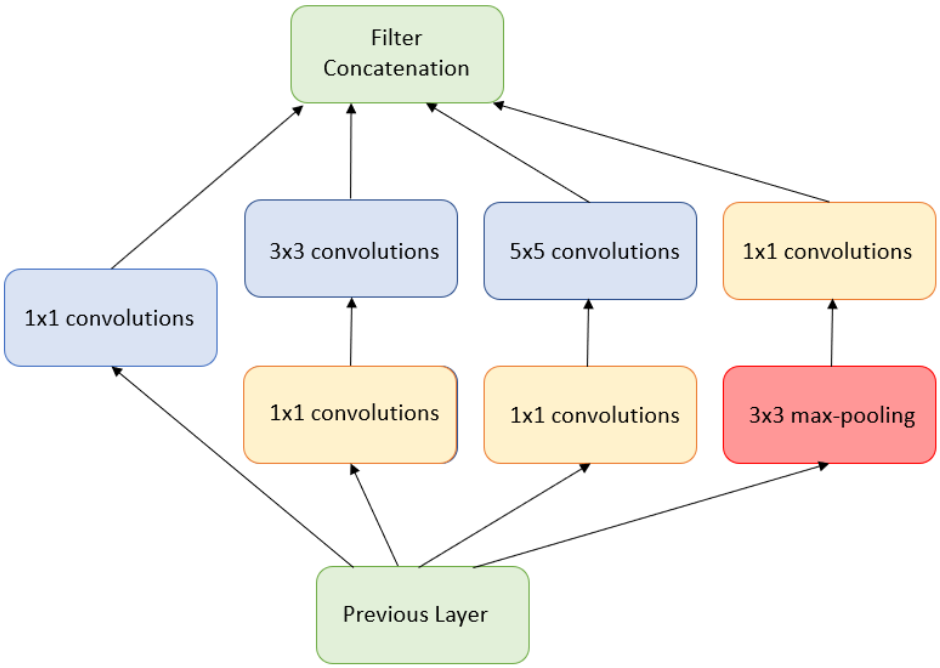


Figure 5-3: Inception Module with dimensionality reduction. The key idea of inception module is to deploy multiple convolutions, with multiple filters and pooling layers simultaneously in parallel within the same layer.

Our Decoder Network tries to regenerate the frame from the hidden representation. Since the input image is used as the target, our Stage-1 training is an unsupervised learning problem. The Decoder takes the hidden representation as input and upsamples it to the original input. Usually, upsampling results are produced by deconvolution operations. However, these deconvolution operations produce checkboard artifacts and hence, we decided to resize using nearest-neighbor interpolation for upsampling and then do a convolution layer.

In general, using an autoencoder means that, at test time, input videos should come from a similar distribution as the training videos, since autoencoders are data specific. However, the person detection and background subtraction applied in the pre-

processing steps make sure that, after preprocessing the data are similarly structured (only upper body visible, no background).

Layer	Kernel	Stride	Padding	In-size	Out-size
Fully-Connected-1				1x7200	30 x 30 x 8
Leaky Relu				30 x 30 x 8	30 x 30 x 8
1Convolution-4	3x3	1x1	SAME	30 x 30 x 8	30 x 30 x 8
Leaky Relu				30 x 30 x 8	30 x 30 x 8
2Upsample-1				30 x 30 x 8	60 x 60 x 8
3Convolution-5	3x3	1x1	SAME	60 x 60 x 8	60 x 60 x 8
Leaky Relu				60 x 60 x 8	60 x 60 x 8
4Upsample-2				60 x 60 x 8	120 x 120 x 8
5Convolution-6	3x3	1x1	SAME	120 x 120 x 8	120 x 120 x 8
Leaky Relu				120 x 120 x 8	120 x 120 x 8
6Upsample-3				120 x 120 x 8	240 x 240 x 8
7Convolution-7	3x3	1x1	SAME	240 x 240 x 8	240 x 240 x 16
Leaky Relu				240 x 240 x 16	240 x 240 x 16
8Logits	3x3	1x1	SAME	240 x 240 x 16	240x240x1
Sigmoid	3			240 x 240 x 1	240x240x1

Table 5-2: Structure of the decoder network. The input and output size are described in *rows x cols x # filters*. The structure uses a nearest-neighbor interpolation up-sampling technique.

This Stage-1 training architecture helps in three ways:

1) It performs dimensionality reduction
2) It captures the spatial information by learning the hidden representation of the frames.
3) It reduces the chances of overfitting when the training data is not large.

The Encoder-Decoder architecture forces the encoder network to learn the most important features in the frames and hence, this makes this architecture best suited for a problem where the number of training data is limited. Using large size networks like [11], [12] increases the chances of overfitting in such cases where the number of training

samples is limited. But the two-stage training is not end-to-end training and hence this makes the training difficult.

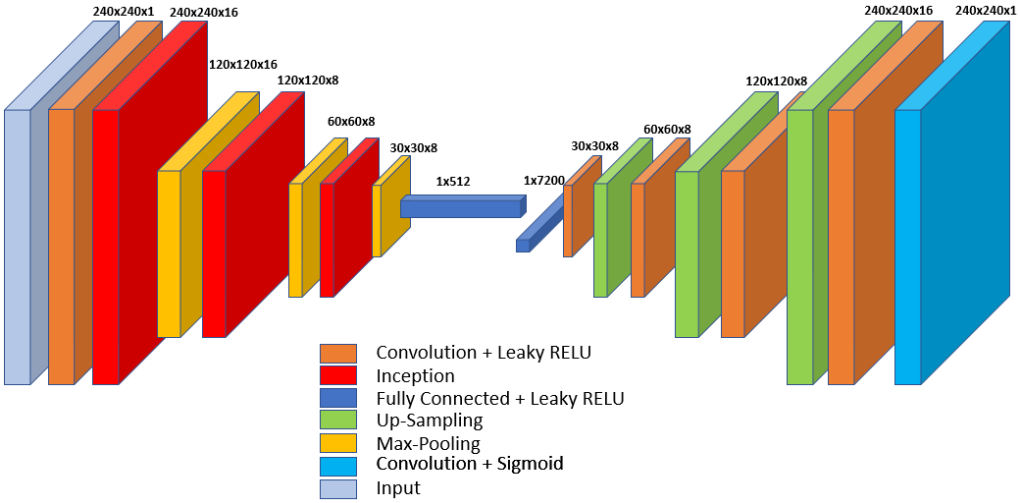


Figure 5-4: Encoder-decoder architecture.

5.2.2 Stage-2 Training Architecture

The stage-2 training uses Model-2 for training to learn the function M: F → S. Model-2 is a LSTM architecture. In the second stage, we experiment with unidirectional LSTM and with bidirectional LSTM. LSTM networks are used to explicitly consider the sequence of encodings coming from Encoder. The LSTM architecture helps in learning temporal information in the videos. The LSTM network uses [13] LSTM cells which learn long-range temporal relationships between the sequences.

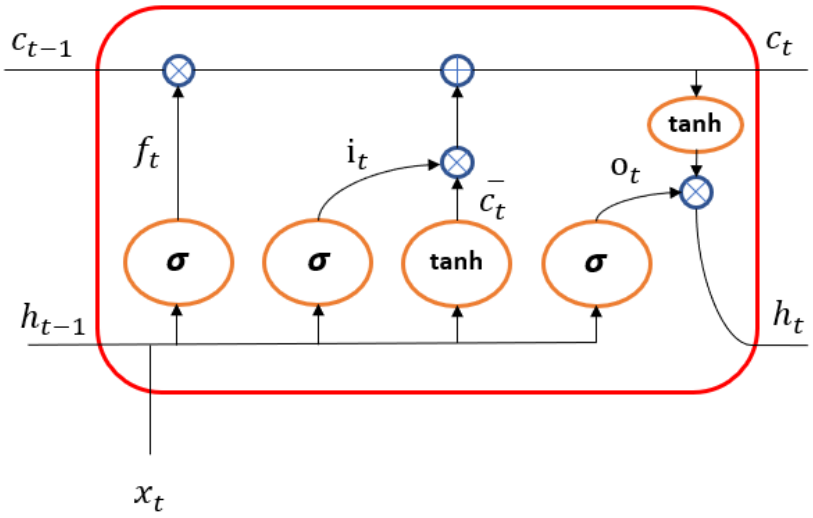


Figure 5-5: LSTM Cell. Each LSTM cell remembers $c_t(\text{Eq. } 5)$. This value may be diminished or erased through a multiplicative interaction with the forget gate $f_t(\text{Eq. } 4)$ or additively modified by the current input x_t multiplied by the activation of the input gate i_t (Eq. 3). The output gate o_t controls the emission of h_t, the stored memory c_t transformed by the hyperbolic tangent nonlinearity (Eq. 6,7).

Given input sequence $x = (x_1, x_2 \dots, x_t)$ a LSTM network computes hidden state sequence $h = (h_1, h_2 \dots h_t)$ and output sequence $y = (y_1, y_2 \dots, y_t)$. The hidden state of a LSTM cell is calculated as follows:

$$i_t = \sigma(W_{xi}x_t + W_{hi}h_t + b_i) \quad (3)$$

$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + b_f\right) \quad (4)$$

$$c_t = f_t c_{t-1} + i_t tanh(W_{xc}x_t + W_{hc}h_{t-1} + b_c) \quad (5)$$

$$o_t = \sigma(W_{xo}x_t + W_{ho}h_{t-1} + b_o) \quad (6)$$

$$h_t = o_t tanh(c_t) \quad (7)$$

where i_t = input gate, f_t = forget gate, o_t = output gate, c_t = cell state, and σ = sigmoidal activation function. The Input gate tells whether to update the current state using the previous state. The Forget gate decides whether to forget the previous hidden state or not. The Cell state keeps only the necessary information. The Output gate filters the emission of the cell state. A unidirectional LSTM layer is a layer with stacked LSTM cells and these layers share weights across time. We freeze the Encoder Network from Stage-1 training and pass the encodings coming from the Encoder Network as input to the Model-2: LSTM architecture. For unidirectional LSTM architecture (See Figure 5-5) we used two layers of LSTM with 128 stacked LSTM cells in each layer which is followed by a SoftMax layer which makes the predictions for every encoded sequence.

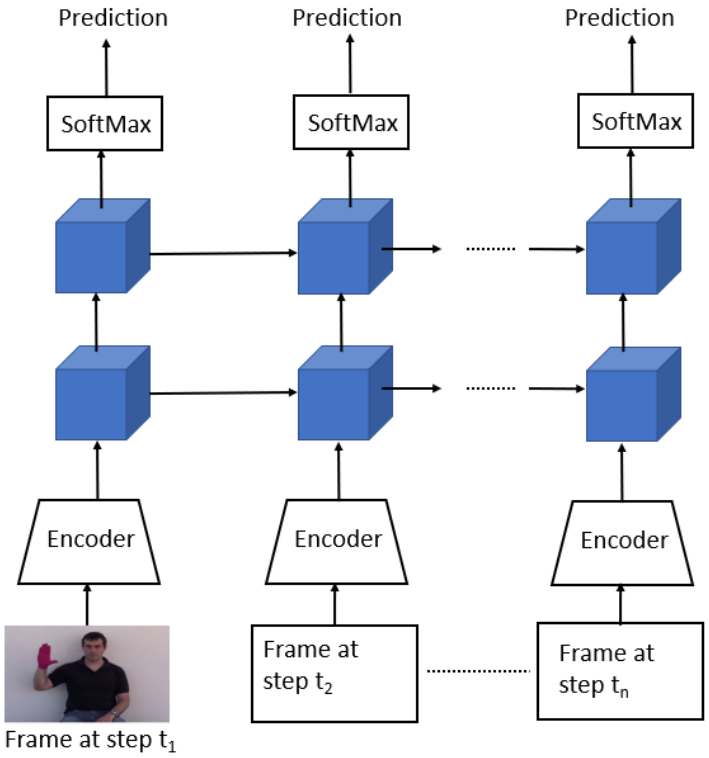


Figure 5-6 Unidirectional LSTM Architecture. Encoder outputs are processed forward through time and upwards through two layers of stacked LSTMs. A SoftMax layer predicts the class at each time step. The parameters of the Encoder network and SoftMax classifier are shared across time steps.

A bidirectional LSTM consists of two LSTM layers, where one layer operates in the forward time direction and the other layer operates in the backward time direction (see Figure 5-6). Because of layers operating in forward and backward direction the bidirectional LSTMs has the advantage over unidirectional LSTMs, the output at each timestep uses information from both the past and the future timesteps. For bidirectional LSTM architecture, we used two layers of bidirectional LSTM with 128 stacked cells in

each layer which is followed by SoftMax layer for making predictions for each encoded sequence.

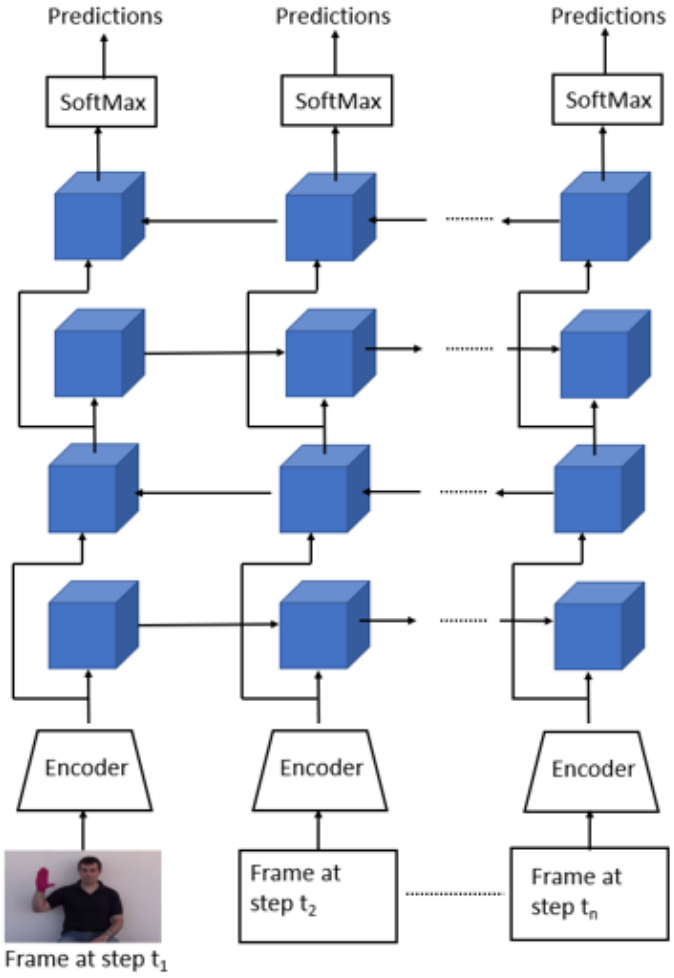


Figure 5-7 Bidirectional LSTM Architecture. Encoder outputs are processed forward through time and upwards through two bidirectional layers of stacked LSTMs. A SoftMax layer predicts the class at each time step. The parameters of the Encoder network and SoftMax classifier are shared across time steps.

These LSTM architectures are trained by backpropagation through time.

Chapter 6

Training and Inference

All the architectures were trained on a Nvidia Geforce GTX 1080 GPU. We used the Tensorflow deep learning framework for out experimentation. Our code is publicly available on Github (https://github.com/jayshah19949596/ASL-Thesis2). Our experimental results show that the bidirectional LSTM architecture performs better than the unidirectional LSTM architecture. We heavily used the Tensorboard tool for visualization and to keep track of inference results. All the Figures presented in this section are taken from Tensorboard results.

6.1 Stage-1 Signer Dependent Training

Stage-1 training is a frame-level training. 30 fps was used to read the video. Every frame was resized to 240x240 which was fed as input to the model. The inputs to the model were the targets and hence stage-1 learning is unsupervised learning. The Encoder-Decoder architecture was trained for 150 epochs. It took approximately 3.5 days to train the model from scratch on 420 videos. The batch size was of varying length. We use mean-squared error as the loss function. We used Adam optimizer as the optimization algorithm with a learning rate of 0.001. There were 280 videos in the test set.

6.2 Stage-1 Signer Dependent Inference

Figure 6-2 shows the batch-loss graph for every batch step. The graph gradually decreased.

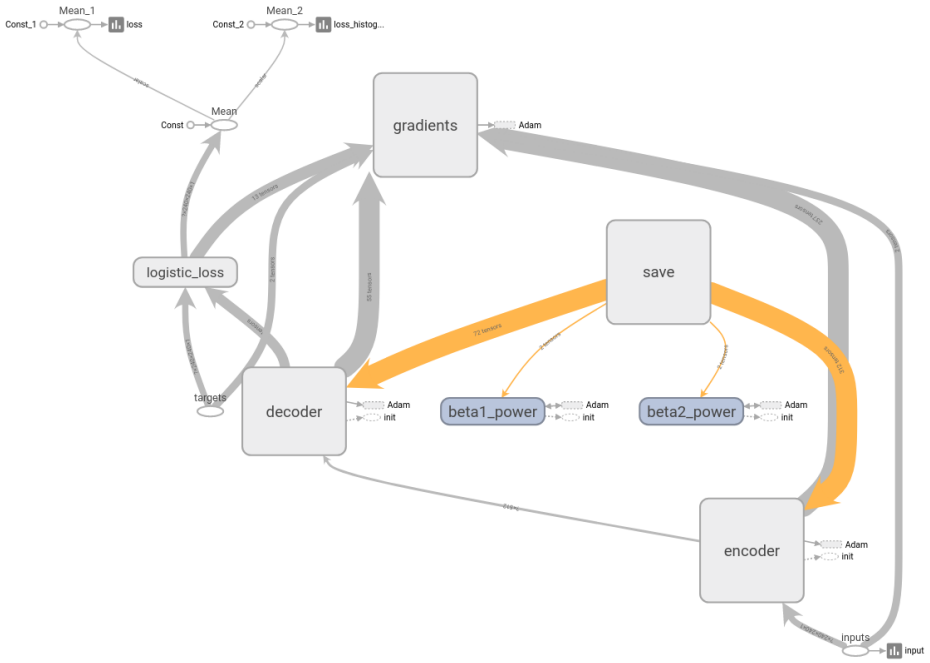


Figure 6-1: Tensorboard signer dependent encoder-decoder main graph.

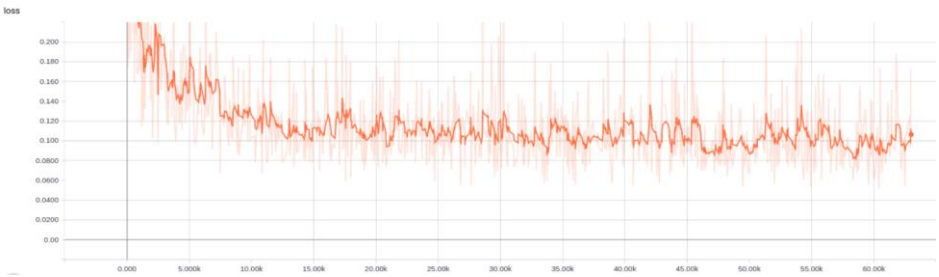
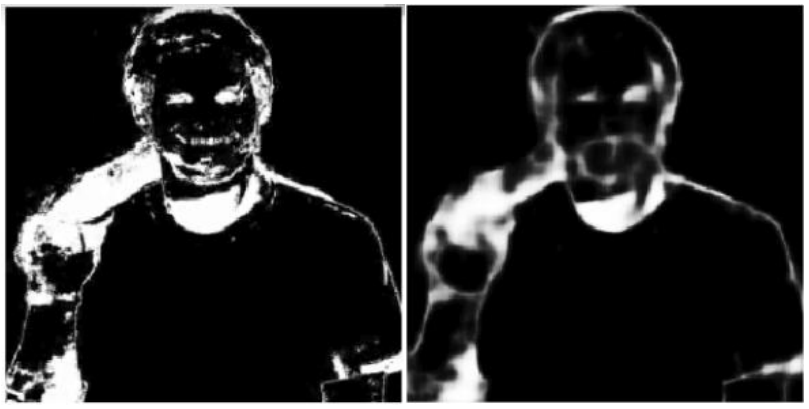


Figure 6-2: Encoder-decoder batch loss graph in stage-1 signer dependent training.



(a)

(b)

Figure 6-3: Stage-1 regeneration results with signer dependent training. Left images are the input to the Encoder-Decoder Network. Right images are the regenerated image by the Encoder-Decoder Network.

6.3 Stage-2 Signer Dependent Training

For the Stage-2 training, the Encoder network is frozen and the Encoder's encoding vector is given as input sequence to the LSTM architecture. Unidirectional LSTM was trained for 60 epochs and bidirectional LSTM model was trained for 50 epochs. For both, it took approximately 1.5 days to complete training. During our

experiments, we found that unidirectional LSTM was overfitting the training data to a certain extent and hence we settled with the model trained for 50 epochs. We used [16] dropout with keep probability of 0.85 for every LSTM layer. Dropout is a regularization technique which reduces overfitting in neural networks by making the signal skip some neurons in the network. We used a batch size of 1 due to GPU memory constraints. The number of time steps used was of varying length for unidirectional LSTM architecture. The maximum number of time steps used for the bidirectional LSTM was 40. For the bidirectional LSTM random sampling was used to extract 40 frames out of the total number of frames if the total number of frames were greater than 40. The loss function used was cross-entropy. We used Adam optimizer as the optimization algorithm, with a learning rate of 0.001.

6.4 Stage-2 Signer Dependent Inference

For both architectures, we used random-cross validation to evaluate validation accuracy during training.

6.4.1 Unidirectional LSTM

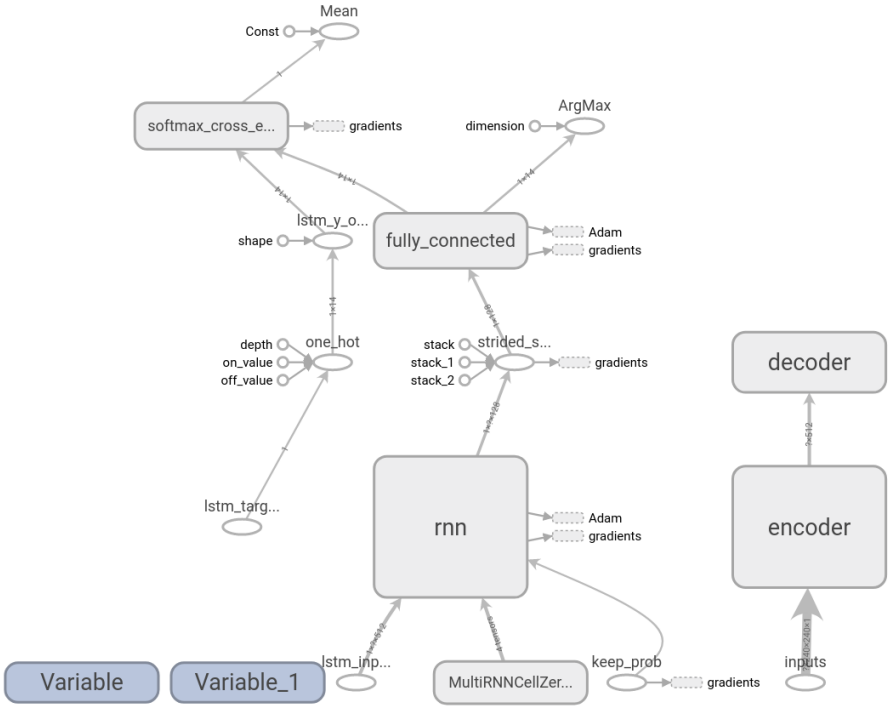


Figure 6-4: Tensorboard signer dependent unidirectional LSTM main graph.

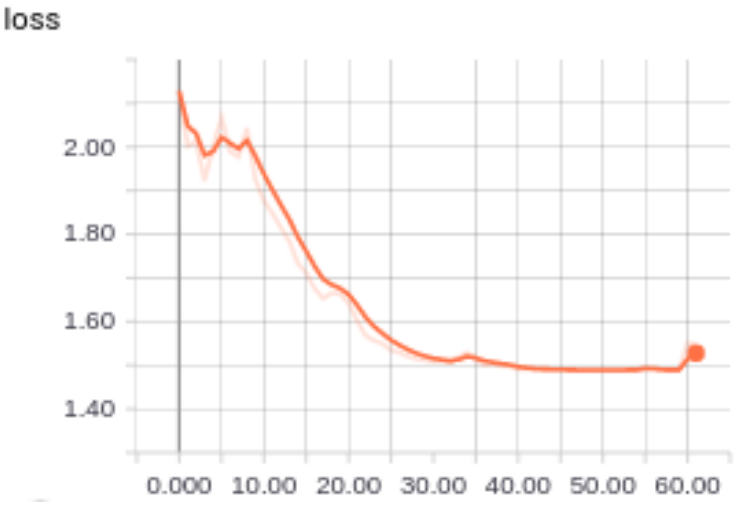


Figure 6-5: Unidirectional LSTM step-loss graph in stage-2 signer dependent training.

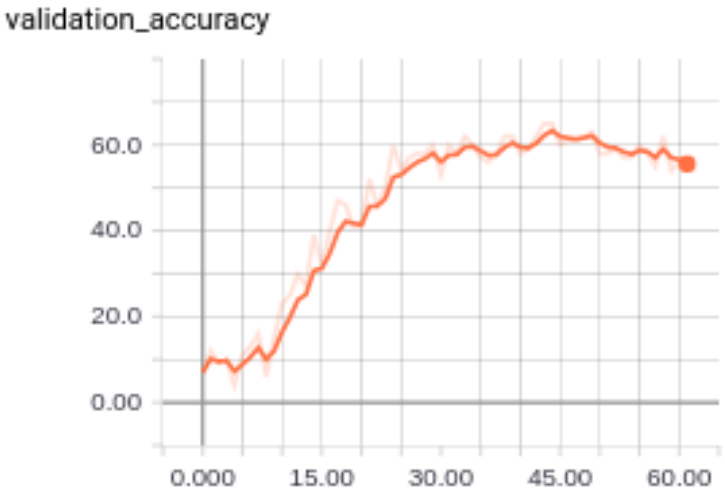


Figure 6-6: Unidirectional LSTM validation accuracy graph in stage-2 signer dependent training.

Table 6-1 shows the evaluation results of Unidirectional LSTM on the test set.

Top – 1	Top – 3	Top – 5
64.28%	90.71 %	96.07 %

Table 6-1: Unidirectional LSTM Signer Dependent Accuracy Table.

6.4.2 Bidirectional LSTM

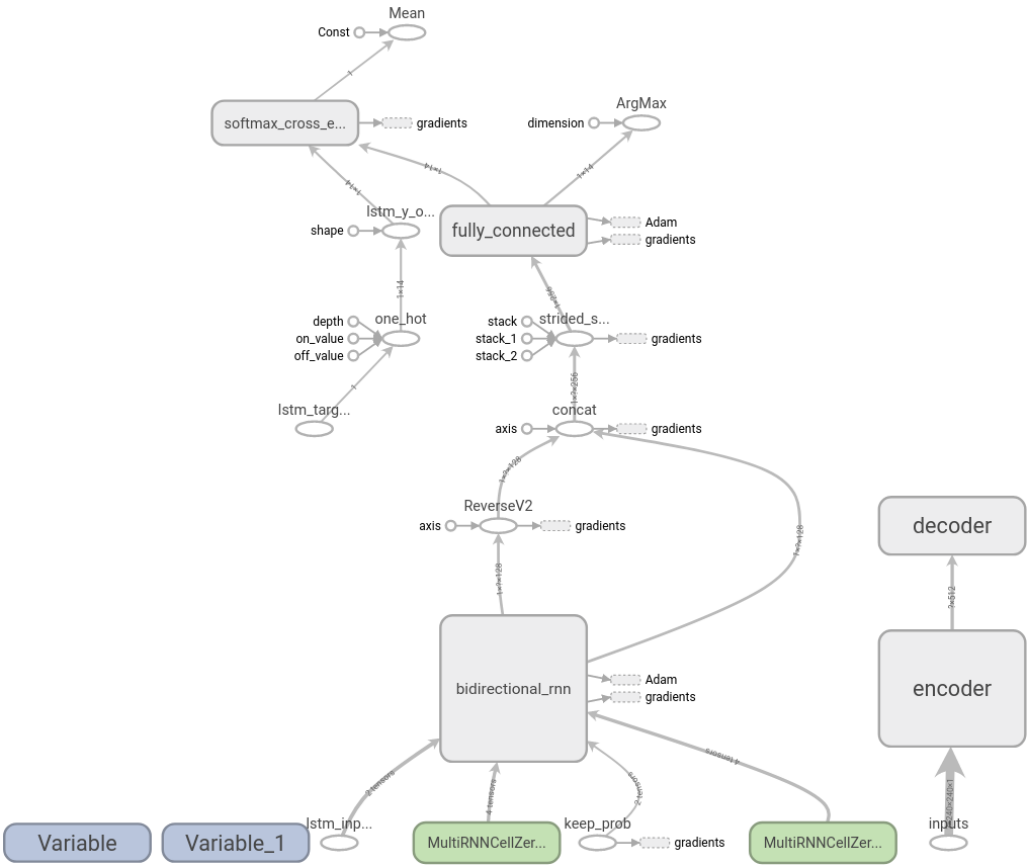


Figure 6-7: Tensorboard signer dependent bidirectional LSTM main graph.

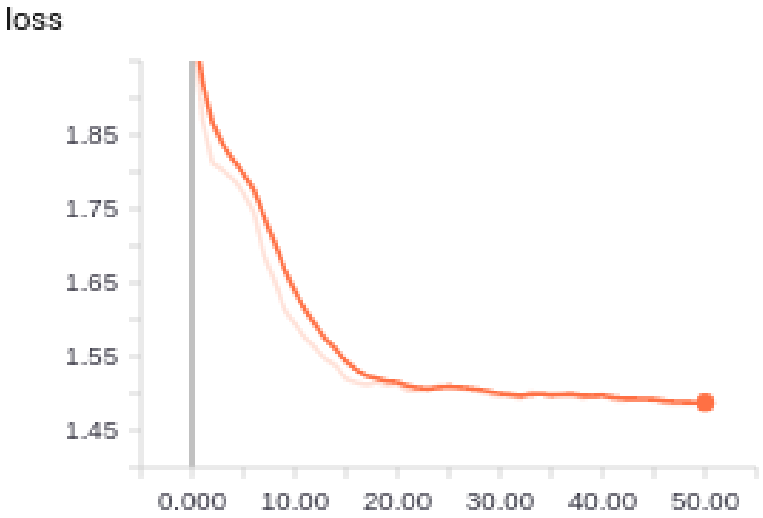


Figure 6-8: Bidirectional LSTM step-loss graph in stage-2 signer dependent training.

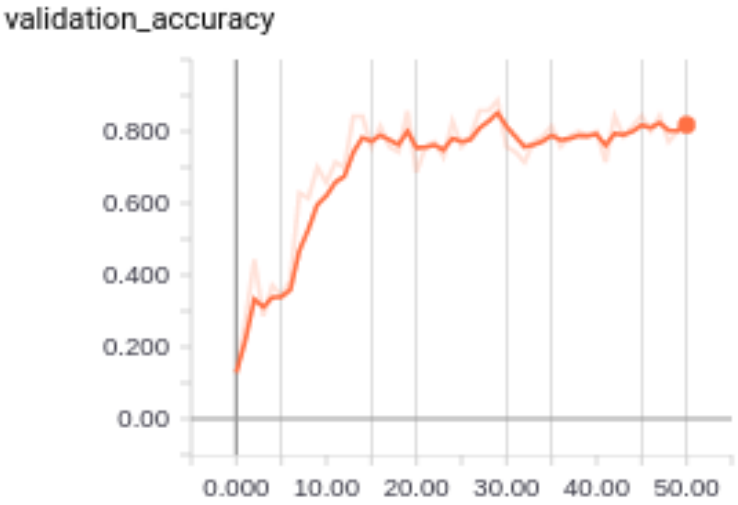


Figure 6-9: Bidirectional LSTM validation accuracy graph in stage-2 signer dependent training.

Top – 1	Top – 3	Top – 5
66.08%	96.02 %	98.20 %

Table 6-2: Bidirectional LSTM Signer Dependent Accuracy Table.

With our experimental results, bidirectional LSTM out-performed unidirectional LSTM. Especially in top 3 predictions, bidirectional LSTM out-performed unidirectional LSTM with a nice margin.

6.5 Stage-1 Signer Independent Training

Stage-1 Signer Independent training used exactly the same settings as the Signer dependent training but the model was trained for more number of epochs.

6.6 Stage-1 Signer Independent Inference

Figure 6-10. shows the batch-loss graph for every batch step. The graph gradually decreased.

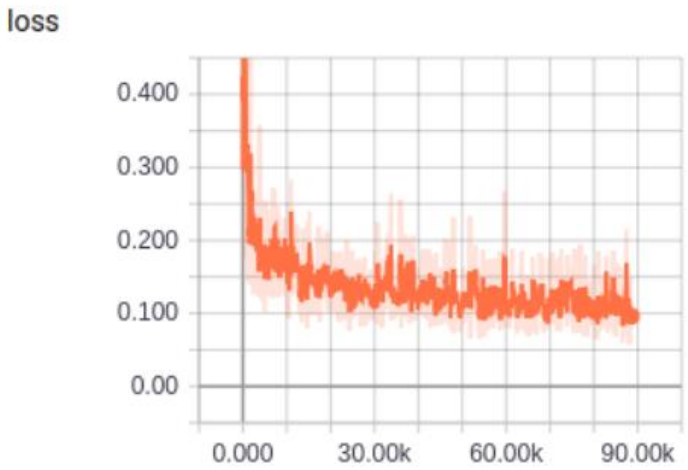


Figure 6-10: Encoder-decoder batch loss graph in stage-1 signer independent training.

(a)

(b)

Figure 6-11: Stage-1 regeneration results with signer independent training. Left Image is the input to the Encoder-Decoder Network. Right side is the regenerated image by the Encoder-Decoder Network.

6.7 Stage-2 Signer Independent Training

For the Stage-2 signer independent training used exactly the same settings as the Stage-2 signer dependent training.

6.8 Stage-2 Signer Independent Inference

For both architectures, we used random-cross validation to evaluate validation accuracy during training.

6.8.1 Unidirectional LSTM

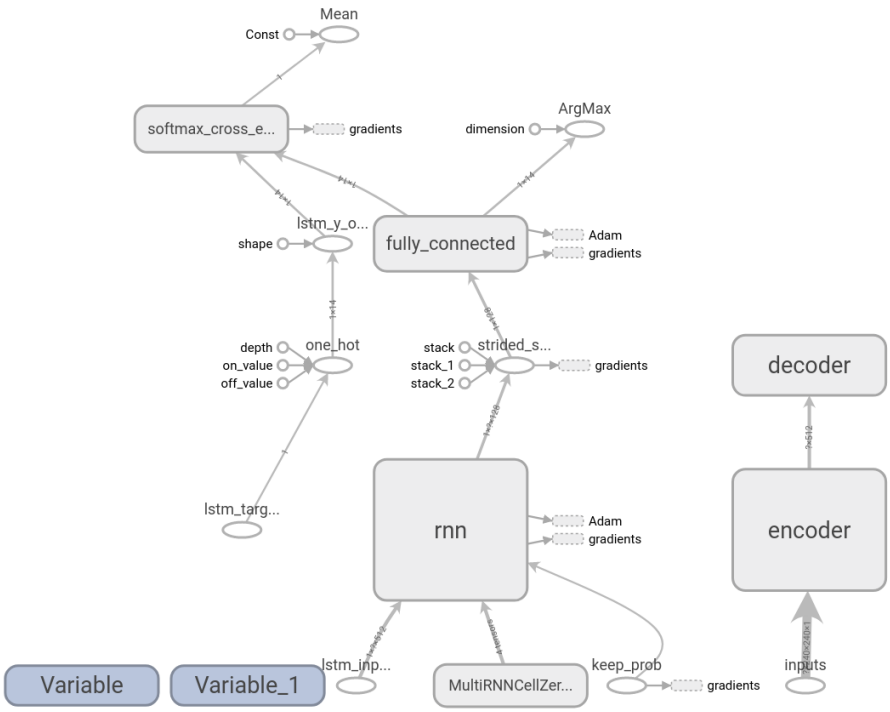


Figure 6-12: Tensorboard signer independent unidirectional LSTM main graph.

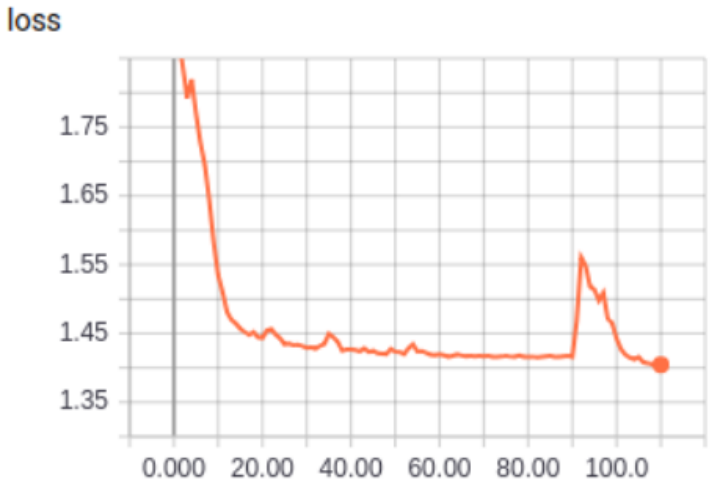


Figure 6-13: Unidirectional LSTM step-loss graph in stage-2 signer independent training.

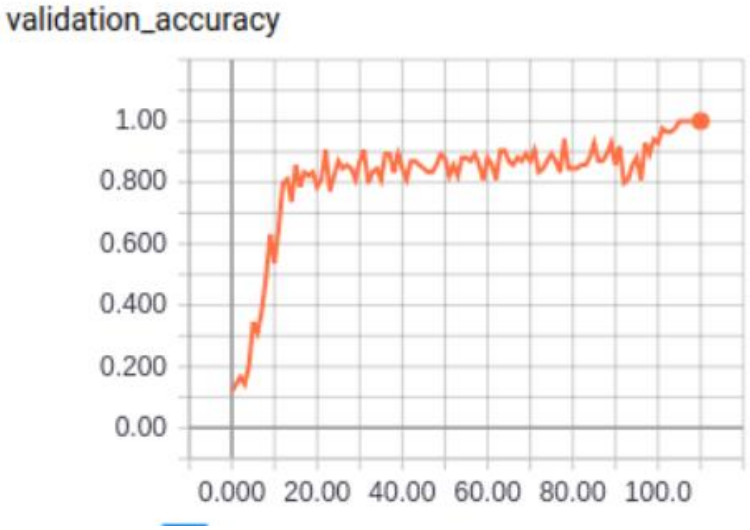


Figure 6-14: Unidirectional LSTM validation accuracy graph in stage-2 signer independent training.

Top – 1	Top – 3	Top – 5
50.35%	71.68 %	81.00 %

Table 6-3: Unidirectional LSTM Signer Independent Accuracy Table

6.8.2 Bidirectional LSTM

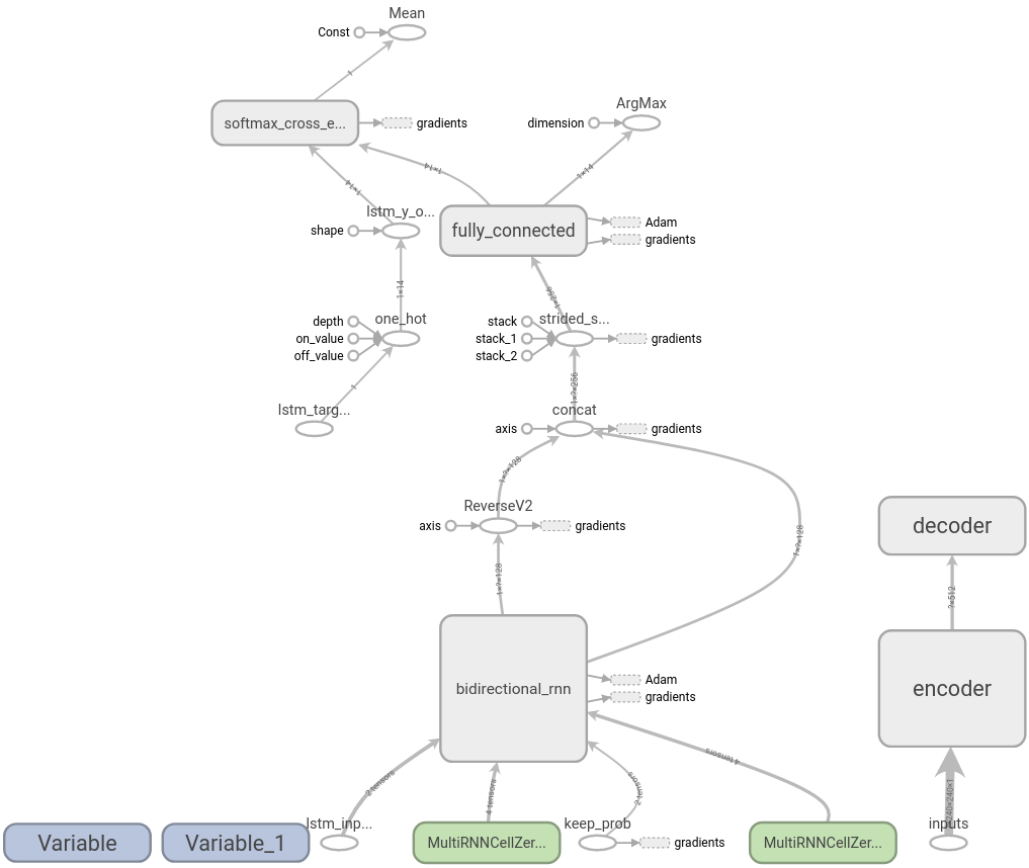


Figure 6-15: Tensorboard signer independent bidirectional LSTM main graph.

Top – 1	Top – 3	Top – 5
57.14%	77.85 %	86.42 %

Table 6-4: Bidirectional LSTM Signer Independent Accuracy Table.

Even in signer independent experiments, bidirectional LSTM out-performed unidirectional LSTM.

6.9 Model performance when scaling data

We also experimented with 20 training samples per class to see how well our model performs when we scale down the data. We used 280 videos for training and 420 videos for testing. We used random sample cross-validation by splitting rule of 70-30.

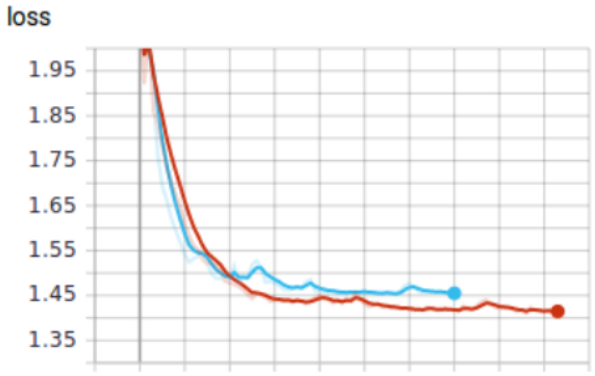


Figure 6-16: Unidirectional LSTM and Bidirectional LSTM step-loss graph in stage-2 signer dependent training with 20 training samples.

Figure 6-17: Unidirectional LSTM and Bidirectional LSTM validation accuracy graph in stage-2 signer dependent training with 20 training samples.

Table 6-5 shows the comparison of the performance of the model with 30 training samples per class and 20 training samples per class with signer dependent settings.

Accuracy / Model	Top - 1	Top - 3	Top - 5	Training samples per class	Epochs
Encoder-BiLSTM	66.08	96.02	98.20	30	50
Encoder-LSTM	64.28	90.71	96.07	30	50
Encoder-BiLSTM	73.88	93.64	97.88	20	90
Encoder-LSTM	60.70	89.71	94.83	20	70

Table 6-5: Accuracy Table when data is scaled.

Chapter 7

Comparing results with DTW

We compare the performance of our models with Dynamic Time Warping approach mentioned in [14]. Dynamic Time Warping is a novel method for Sign Language Recognition when you have limited training data. The hand co-ordinates and the face co-ordinates required in Dynamic Time Warping were obtained from [15] OpenPose framework.

7.1 Dynamic Time Warping

Dynamic Time Warping (DTW) is a novel method which is used for measuring the similarity between two sequences of different lengths. DTW finds similarity between sequences by computing a distance score.

7.1.1 Computing the Similarity Score

Let *X* be a sign video which is a sequence of frames $(x_1, x_2, \dots, x_n)$. For every frame x_t at timestamp ***t*** compute its corresponding feature $f_t(x_t)$ at timestamp ***t*** which is a function of x_t. The feature f_t at timestamp ***t*** is the concatenation of two features:

$$f_t(x_t) = [L_d(x_t), O_d(x_t)] \qquad (8)$$

Where,

$L_d(x_t)$ = The (x, y) wrist location of the signer's hand at frame ***t***

$O_d(x_t)$ = The unit vector which is direction of motion form $L_d(x_{t-1})$ to $L_d(x_{t+1})$

Given two video *X* and *Y* DTW a warping path *W* between the feature of *X* and *Y:*

$$W = ((f_1(x_1), f_1(y_1)), \dots, (f_n(x_n), f_n(y_n))) \qquad (9)$$

The distance score $D(W, X, Y)$ of a warping path W is the sum of individual distance scores $\mathrm{d}\left(\mathrm{X}_{\mathrm{f_i(x_i)}}, \mathrm{Y}_{\mathrm{f_i(y_i)}}\right)$

$$D(W,X,Y) = \sum_{i=1}^{n} d\left(X_{f_i(x_i)}, Y_{f_i(y_i)}\right) \quad (10)$$

The distance scored is a weighted linear combination of individual Euclidean distances between the features extracted from frames:

$$d\left(X_{f_i(x_i)}, Y_{f_i(y_i)}\right) = ||L_d(x_t) - L_d(y_t)|| + ||O_d(x_t) - O_d(y_t)|| \quad (11)$$

where ||·|| stands for Euclidean Distance.

The DTW distance $DTW(X,Y)$between sign videos *X* and *Y* is defined as the cost of the lowest-cost warping path between *X* and *Y*:

$$DTW(X,Y) = min_W D(W,X,Y) \quad (12)$$

7.1.2 Feature extraction with OpenPose

OpenPose was used for extracting the $L_d(x_t)$ feature.

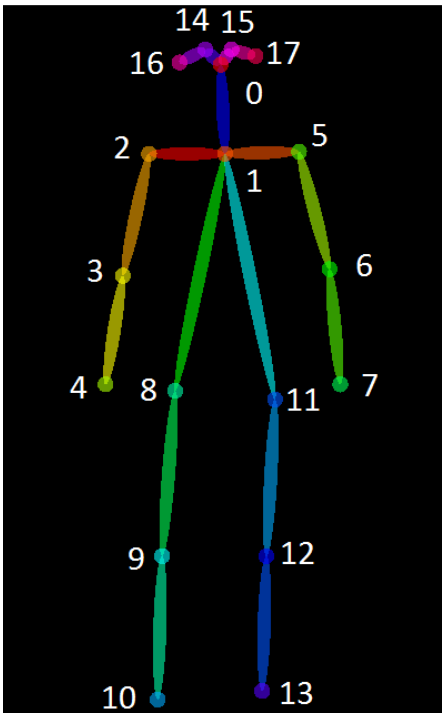


Figure 7-1: OpenPose pose estimation output.

OpenPose stores the coordinates of the key-points in a JSON format. The coordinate of the 4th key-point represents $L_d(x_t)$ for that frame.

7.2 Comparing all models

Accuracy / Method	Top - 1	Top - 3	Top - 5	Signer Independent	Epochs
Encoder with Bi-LSTM	66.08	96.02	98.20	No	50
Encoder with LSTM	64.28	90.71	96.07	No	50
DTW	58.21	86.64	93.21	No	-
Encoder with Bi-LSTM	57.14	77.85	86.42	Yes	120
Encoder with LSTM	50.35	71.68	81.00	Yes	120
Encoder-BiLSTM	73.88	93.64	97.88	No	20
Encoder-LSTM	60.70	89.71	94.83	No	20

Table 7-1: Accuracy Table or all models.

Table 7-1 shows comparative results for the five methods where accuracy is the quantitative measure used to evaluate the individual method and compare with other methods. Our both models clearly outperform DTW in all the cases.

Chapter 8

Discussion & Future Work

We introduced a new deep learning architecture called DeepSign for Sign Language Recognition. DeepSign uses Encoder Decoder Architecture to extract important and unique features which is helpful in the classification. Deep Learning architectures when designed carefully, can outperform traditional methods on a dataset limited training data.

We can incorporate more sophisticated techniques like visual attention, hand shape recognition, hand detection in the model to improve the results. Currently, the model gets confused between the sign having the same motion so by techniques like visual attention, hand shape recognition, hand detection we can make the model distinguish between the two signs with the same motion.

References

[1] Thad Starner and Alex Pentland “ Real-Time American Sign Language Recognition from Video Using Hidden Markov Models” ISCV '95 Proceedings of the International Symposium on Computer Vision.

[2] Kalpattu S. Abhishek, Lee Chun Fai Qubeley and Derek Ho “Glove-based hand gesture recognition sign language translator using capacitive touch sensor” 2016 IEEE International Conference on Electron Devices and Solid-State Circuits (EDSSC).

[3] M. P. Paulraj, Sazali Yaacob, Hazry Desa, C.R. Hema, Wan Mohd Ridzuan, Wan Ab Majid “Extraction of head and hand gesture features for recognition of sign language” 2008 International Conference on Electronic Design.

[4] Sriparna Saha, Rimita Lahiri, Amit Konar, Atulya K. Nagar “A novel approach to American sign language recognition using MAdaline neural network” 2016 IEEE Symposium Series on Computational Intelligence (SSCI).

[5] Bowen Shi, Karen Livescu “Multitask training with unlabeled data for end-to-end sign language fingerspelling recognition”.

[6] Brandon Garcia and Sigberto Alarcon Viesca “Real-time American Sign Language Recognition with Convolutional Neural Networks”.

[7] Oscar Koller, Sepehr Zargaran, Hermann Ney and Richard Bowden “Deep Sign: Hybrid CNN-HMM for Continuous Sign Language Recognition” In British Machine Vision Conference (BMVC), York, UK, September 2016.

[8] Pavlo Molchanov, Xiaodong Yang, Shalini Gupta, Kihwan Kim, Stephen Tyree and Jan Kautz “Online Detection and Classification of Dynamic Hand Gestures with Recurrent 3D Convolutional Neural Networks” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR).

[9] Ronchetti, Franco and Quiroga, Facundo and Estrebou, Cesar and Lanzarini, Laura and Rosete, Alejandro “LSA64: A Dataset of Argentinian Sign Language” in 2016 XX II Congreso Argentino de Ciencias de la Computación (CACIC).

[10] "Speed/accuracy trade-offs for modern convolutional object detectors." Huang J, Rathod V, Sun C, Zhu M, Korattikara A, Fathi A, Fischer I, Wojna Z, Song Y, Guadarrama S, Murphy K, CVPR 2017.

[11] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, Andrew Rabinovich “Going Deeper with Convolutions” in CVPR 2015.

[12] Karen Simonyan and Andrew Zisserman “VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION” in ICLR 2015.

[13] Sepp Hochreiter and Jurgen Schmidhuber “LONG SHORT-TERM MEMORY”.

[14] H. Wang, A. Stefan, S. Moradi, V. Athitsos, C. Neidle, and F.Kamangar. “A System for Large Vocabulary Sign Search," in Workshop on Sign, Gesture and Activity (SGA), September 2010, pp. 1-12.

[15] Zhe Cao and Tomas Simon and Shih-En Wei and Yaser Sheikh “Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields” in CVPR 2017.

[16] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, Ruslan Salakhutdinov “Dropout: A Simple Way to Prevent Neural Networks from Overfitting” in Journal of Machine Learning Research 15 (2014).

Biographical Information

Jai Shah was born in Maharashtra, India in 1994. He has received his B.E in Information Technology from Mumbai University, Mumbai, Maharashtra, India, in 2016, his M.S. degree in Computer Science from The University of Texas at Arlington, Arlington, Texas, USA in 2018. He has worked as an Android Developer intern at Unum Inc., Los Angeles, California, USA in 2017. He will be joining Charles Schwab, Westlake, Texas, USA in 2018 as a Software Engineer.