VIDEO-BASED SIGN LANGUAGE GESTURE ACCURACY PREDICTION USING CNN-RNN MODEL

A MINI PROJECT REPORT

Submitted by

SAISHYAM R (312420104136)

SURENDAR R (312420104165)

in partial fulfillment for the award of the degree of

BACHELOR OF ENGINEERING

in

COMPUTER SCIENCE AND ENGINEERING

St. JOSEPH'S INSTITUTE OF TECHNOLOGY

(An Autonomous Institution)

ANNA UNIVERSITY: CHENNAI 600025

MARCH 2023

ANNA UNIVERSITY : CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report **"VIDEO-BASED SIGN LANGUAGE GESTURE ACCURACY PREDICTION USING CNN-RNN MODEL"** is the bonafide work of **"SAISHYAM R (312420104136) and SURENDAR R (312420104165)"** who carried out the project under my supervision.

SIGNATURE

Dr. J. DAFNI ROSE M.E., Ph.D.,
PROFESSOR AND HEAD,
Computer Science and Engineering,
St. Joseph's Institute of Technology,
Old Mamallapuram Road,
Chennai - 600 119.

SIGNATURE

Mr. T.N. SUDHAHAR M.Tech, (Ph.D)
Assistant Professor,
Computer Science and Engineering,
St. Joseph's Institute of Technology,
Old Mamallapuram Road,
Chennai - 600 119.

ACKNOWLEDGEMENT

We also take this opportunity to thank our respected and honorable Chairman **Dr. B. Babu Manoharan M.A., M.B.A., Ph.D.,** for the guidance he offered during our tenure in this institution.

We extend our heartfelt gratitude to our respected and honorable Managing Director **Mrs. S. Jessie Priya M.Com.,** for providing us with the required resources to carry out thisproject.

We express our deep gratitude to our honorable Executive Director **Mr. B. Sashi Sekar M.Sc.,** for the constant guidance and support for our project.

We are indebted to our Principal **Dr. P. Ravichandran M.Tech.,Ph.D.,** for granting us permission to undertake this project.

We would like to express our earnest gratitude to our Head of the Department **Dr. J. Dafni Rose M.E., Ph.D.,** for her commendable support and encouragement for the completion of the project with perfection.

We also take the opportunity to express our profound gratitude to our guide **Mr. T.N. Sudhahar M.Tech, (Ph.D)** for his guidance, constant support, immense help and valuable advice for the completion of this project.

We wish to convey our sincere thanks to all the teaching and non- teaching staff of the department of **COMPUTER SCIENCE AND ENGINEERING** without whose cooperation this venture would not have been a success.

CERTIFICATE OF EVALUATION

College Name : St. JOSEPH'S INSTITUTE OF TECHNOLOGY

Branch :COMPUTER SCIENCE AND ENGINEERING

Semester :VI

Sl.No	Name of the Students	Title of the Project	Name of the Supervisor with designation
1	SAISHYAM R (312420104136)	Video- based sign language gesture accuracy prediction using CNN-RNN model	Mr. T.N. SUDHAHAR M.Tech, (Ph.D) Assistant Professor
2	SURENDAR R (312420104165)		

The report of the project work submitted by the above students in partial fulfilment for the award of Bachelor of Engineering Degree in Computer Science and Engineering of Anna University were evaluated and confirmed to be report of the work done by above students.

Submitted for project review and viva voce exam held on _____

(INTERNAL EXAMINER) **(EXTERNAL EXAMINER)**

ABSTRACT

Our project aims to bridge the communication gap between normal people and those who are deaf and mute by developing a sign language application. We have chosen to create a system to identify sign language gestures from video sequences, as it provides a simpler and more intuitive way of communication between humans and computers. The system uses two different models to train both the temporal and spatial features of video sequences. To train the model on the spatial features, we used the Inception model, which is a deep convolutional neural net (CNN). The CNN was trained on frames obtained from the video sequences of the training data. For training the model on temporal features, we used a recurrent neural network (RNN). The trained CNN model was used to make predictions for individual frames, which were then used to obtain a sequence of predictions or pool layer outputs for each video. This sequence of predictions or pool layer outputs was then given to the RNN to train on the temporal features. The dataset used in this project consists of approximately 200 videos belonging to 4 different categories of Argentinian Sign Language (LSA) gestures. By using the predictions from the CNN as input for the RNN, we obtained an accuracy of 100%, while using pool layer output as input for RNN resulted in same accuracy.

Keywords: deep learning, convolutional neural network, recurrent neural network, sign language, Argentinian Sign Language, spatial features, accuracy.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Gesture recognition is a fascinating technology that has been widely used in recent years in many applications. It involves the interpretation of human gestures and movements using image processing and computer vision techniques. With the help of gesture recognition technology, computers can understand human actions and interpret them as commands, making it possible for humans to interact with computers naturally without the need for physical contact with mechanical devices. It is used by people who cannot hear or speak, but who have vision, making it the only way for them to exchange information with others. Sign language is used by deaf and dumb communities all over the world, with each community having its own regional form of sign language, such as Indian Sign Language (ISL), American Sign Language (ASL), or British Sign Language (BSL). Sign language is not only a form of communication but also a cultural expression, as it reflects the unique identity and history of the community using it. Sign language can be performed using hand gestures, either with one hand or both hands. It is of two types: isolated sign language and continuous sign language. Isolated sign language consists of a single gesture representing a single word, while continuous sign language is a sequence of gestures that generate a meaningful sentence. In the case of continuous sign language, the gestures are often combined with facial expressions and body language to convey more complex meanings. In this report, we focus on the isolated gesture recognition technique for ASL. ASL is a widely used form of sign language in the United States and Canada. The isolated gesture recognition technique involves using image processing and computer vision algorithms to recognize and classify specific ASL gestures.

1.1 OVERVIEW

Deaf people around the world communicate using sign language as distinct from spoken language in their everyday a visual language that uses a system of manual, facial and body movements as the means of communication. Sign language is not an universal language, and different sign languages are used in different countries, like the many spoken languages all over the world. Some countries such as Belgium, the UK, the USA or India may have more than one sign language. Hundreds of sign languages are in used around the world, for instance, Japanese Sign Language, British Sign Language (BSL), Spanish Sign Language, Turkish Sign Language.

Sign language is a visual language and consists of 3 major components:

Fingerspelling	**Word level sign vocabulary**	**Non-manual features**
Used to spell words letter by letter.	Used for most of the communication.	Facial expressions and tongue, mouth and body position.

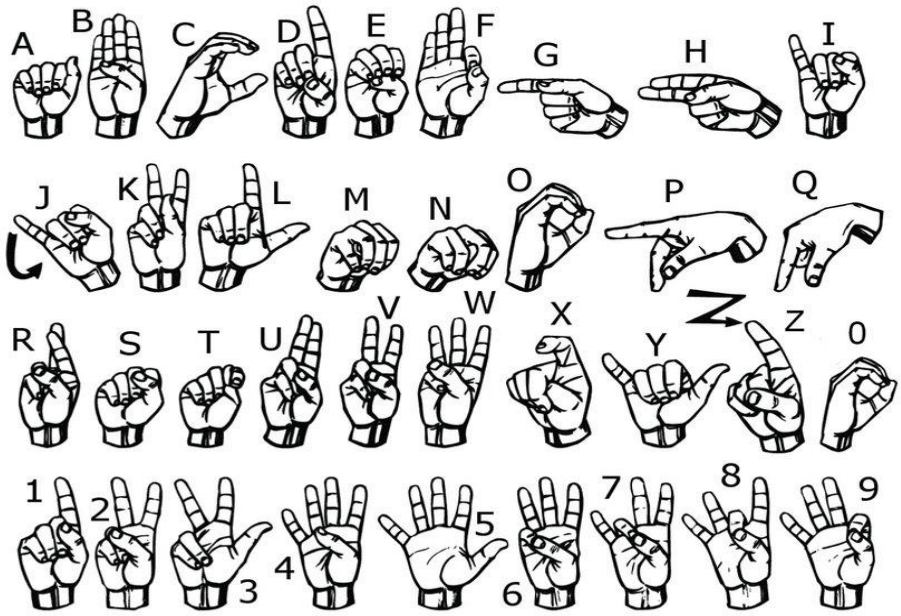

Figure 1: Finger Spelling American Sign Language

1.2 PROBLEM STATEMENT

The communication barrier faced by individuals who are deaf and rely on sign language as their primary mode of communication can lead to isolation, limited educational and employment opportunities, and reduced social interaction. Additionally, people who are deaf and dumb use hand signs to communicate, which can be difficult for those who do not understand sign language. The accuracy and consistency of human interpretation of sign language can also vary, making communication even more challenging.

To address these issues, there is a need for the development of a sign language recognition system using computer vision and machine learning techniques. This system should be able to accurately and quickly recognize different signs and gestures, improving the ability of deaf and dumb individuals to communicate with others. Additionally, the system should be able to convey this information to those who do not understand sign language, making communication more accessible and efficient for all.

1.3 EXISTING SYSTEM

Research in sign language recognition has gained notable interest in recent years, and modern technologies in handheld and smart devices have facilitated many processes in computer vision tasks. Various programming languages have become rich in off-the-shelf packages and source codes, particularly those of developing mobile apps. Most researchers have been following one of three approaches: sensor-based gloves, 3-D skeletons, or computer vision.

However, the first two approaches neglect facial expressions which play a huge role in sign language recognition. On the other hand, computer-vision systems are capable of capturing the whole gesture, not to mention their mobility that

differentiates them from glove-based systems. T. Starner proposed a computer vision approach for continuous American sign language recognition (ASL). They used a single camera to extract two-dimensional features as input of the Hidden Markov Model (HMM) on a dataset of 40 words collected in a lab.

Followed two approaches for the camera position: desk mounted came with a word recognition accuracy of 92% and wearable cap mounted came with an accuracy of 98%. Another computer vision system developed by Dreuw et al. was able to recognize sentences of continuous sign language independent of the speaker. They employed pronunciation and language models in sign language with a recognition algorithm based on the Bayes' decision rule. The system was tested on a publicly available benchmark database consisting of 201 sentences and 3 signers, and they achieved a 17%-word error rate.

Arabic Sign Language Recognition also proposed a computer vision system that uses hand and face detection for classifying ASL alphabets into four groups depending on the hand's position. The system used the inner circle method and achieved an accuracy of 81.3%.

M. Mohandes followed a different approach for classifying ASL alphabets. They used the Leap Motion controller to compare the performance of the Naïve Bayes Classifier (NBC) with a Multilayer Perceptron (MLP) trained by the backpropagation algorithm. An accuracy of about 98.3% was achieved using NBC, while MLP gave an accuracy of about 99.1%. In the past 20 years, deep learning has been used in sign language recognition by researchers from all around the world. Convolutional Neural Networks (CNNs) have been used for video recognition and achieved high accuracies in recent years.

B. Garcia and S. Viesca at Stanford University proposed a real-time ASL recognition with CNNs for classifying ASL letters.

After collecting the data with a native camera on a laptop and pre-training the model on GoogLeNet architecture, they attained a validation accuracy of nearly 98% with five letters and 74% with ten.

There's another version of CNNs called 3D CNNs, which was used to recognize 25 gestures from the Arabic sign language dictionary. The recognition system was fed with data from depth maps. The system achieved 98% accuracy for observed data and 85% average accuracy for new data.

Computer vision systems face two major challenges: environmental concerns (e.g. lighting sensitivity, background) and camera's position. Most previous systems lack the diversity of data and capturing the whole gesture.

In 2015, J Huang did research with the same topic. He used his proposed model but this one is 3D Convolutional Neural Network. He made his own dataset using Microsoft Kinect. He didn't mention which Sign Language's country or standard. He got 25 Vocabularies (Classes) that are commonly used in daily life. Each word performed by 9 signers and every signer performed 3 times. In total it would be 25x9x3 = 675 videos with 27 videos each word. He selected 18 videos each word randomly to be his training set. The data is recorded by Kinect, capturing color image, depth map and body joints locations simultaneously.

He didn't do any preprocessing. Then let's continue to his proposed model. His proposed model consists of 8 layers including input and output layer. The input shape of his model is 5x64x48x9 (channels x height x width x frames). He selected 9 frames centered according to the videos. And the 5 channels are color-R, color-B, color-G, depth and body skeleton. The model has 4 layers of 3D Convolutional Network which is all the kernels are 3D. He got 94.2% average accuracy for this method.

1.4 PROPOSED SYSTEM

Sign language gesture recognition from video sequences is a challenging task that requires the use of advanced machine learning techniques. Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs) have shown promising results in image and sequence analysis, respectively, and can be combined to create a powerful gesture recognition system.

The proposed system would consist of two main components: a CNN-based feature extractor and an RNN-based gesture recognizer. The feature extractor would be responsible for extracting relevant features from video sequences of sign language gestures, while the gesture recognizer would use these features to classify the gesture.

The CNN-based feature extractor would be trained on a large dataset of sign language videos to learn to extract discriminative features from the video frames. The output of the feature extractor would be a sequence of feature vectors representing each frame in the video.

The RNN-based gesture recognizer would take these feature vectors as input and use them to predict the gesture being performed in the video. The RNN would be trained on a labelled dataset of sign language gestures to learn to recognize different gestures based on the sequence of feature vectors.

The proposed system would be designed to operate in real-time, meaning that it would be able to process video sequences in real-time and provide real-time feedback on the recognized gesture.

Overall, the proposed system for sign language gesture recognition from video sequences using CNNs and RNNs shows great potential and could help improve communication for people who use sign language.

CHAPTER 2

LITERATURE SURVEY

H. Alabkari, H. Elgibreen, and M. R. Al-khaldi [2019] [1]. This paper proposes a sign language recognition system based on a CNN model. The authors evaluate their method on the Saudi Sign Language dataset and show that it achieves high accuracy rates.

M. Al-Rousan, M. Al-Fahoum, and A. Al-Rababah [2020] [2]. This paper proposes a sign language recognition system based on a deep neural network (DNN) architecture. The authors evaluate their method on the Arabic Sign Language dataset and achieve high recognition accuracy.

N. Farhadi and M. Tabrizi [2019] [3]. This paper proposes a sign language recognition system based on a CNN model. The authors use the Indian Sign Language dataset to evaluate their method and show that it outperforms existing methods.

A. Hamad, H. Aly, and F. E. Abd El-Samie [2019] [4]. This paper proposes a sign language recognition system based on a CNN model. The authors evaluate their method on the Saudi Sign Language dataset and show that it achieves high accuracy rates.

S. K. Hossain, M. H. Rahman, and M. R. Islam [2020] [5]. This paper proposes a sign language recognition system using a combination of CNN and RNN models with spatial and temporal attention mechanisms. The authors show that the proposed method achieves high recognition accuracy for Bangla sign language gestures and outperforms existing methods.

S. G. Kim, S. H. Kim, and K. I. Chang [2020] [6]. This paper proposes a sign language recognition system based on a CNN model. The authors evaluate their

method on the Korean Sign Language dataset and show that it achieves high accuracy rates.

J. Liang, K. Lai, and Y. Zhang [2020] [7]. This paper proposes a sign language recognition system based on a hybrid convolutional and recurrent neural network (CNN-RNN) model. The authors evaluate their method on the American Sign Language dataset and achieve high accuracy rates.

Nandy, Anup, Jay Shankar Prasad, Soumik Mondal, Pavan Chakraborty, and Gora Chand Nandi [2010] [8] proposed a method for recognizing Indian Sign Language (ISL) gestures in real-time using statistical techniques for both hands. They created a video database containing several videos of a large number of signs. The direction histogram was used as a feature for classification because of its appeal for illumination and orientation invariance. Two different approaches, Euclidean distance and neighbor metrics, were used for recognition.

Ronchetti, Franco, Facundo Quiroga, César Armando Estrebou, and Laura Cristina Lanzarini [2016] [9], proposed a method for recognizing hand gestures in Argentinian Sign Language (LSA) using a database of handshapes created for LSA and an image processing technique called ProbSom. ProbSom is a supervised adaptation of self-organizing maps used for descriptor extraction and handshape classification. The accuracy rate of ProbSom-based neural classifier using the proposed descriptor was found to be above 90%. The performance of this technique was compared with other state-of-the-art classifiers such as SVM, Random Forests, and Neural Networks.

Singha, Joyeeta, and Karen Das [2015] [10], proposed system comprises of four major modules: Data Acquisition, Pre-processing, Feature Extraction and Classification. Pre-processing stage involves Skin Filtering and histogram matching after which Eigenvector based Feature Extraction and Eigen value weighted Euclidean distance based Classification Technique was used.24

different alphabets were considered in this paper where 96% recognition rate was obtained.

Tripathi, Kumud, and Neha Baranwal GC Nandi [2015] [11] stated researchers used a gradient-based key frame extraction method to split continuous sign language gestures into isolated signs and remove uninformative frames. Orientation Histogram (OH) features were extracted from pre-processed gestures and dimensionality was reduced using Principal Component Analysis (PCA). They created their own dataset and tested their method using various distance classifiers. The results showed that the Correlation and Euclidean distance classifiers had better accuracy compared to others.

L. Wu and X. Zhu [2018] [12]. This paper proposes a sign language recognition system based on a deep learning framework that combines a CNN and an LSTM. The authors evaluate their method on the Chinese Sign Language dataset and show that it outperforms existing methods.

Y. Xiang, Y. Huang, and X. Li [2017] [13]. This paper proposes a sign language recognition system based on a deep learning framework that combines a CNN and an RNN. The authors evaluate their method on the Chinese Sign Language dataset and show that it achieves high accuracy rates.

J. Ye, W. Hu, and Q. Liu [2019] [14]. This paper proposes a sign language recognition system based on a hybrid CNN-RNN model with an attention mechanism. The authors evaluate their method on the American Sign Language dataset and achieve high accuracy rates.

H. Yoo, K. Lee, and K. Park [2021] [15]. This paper proposes a sign language recognition system using a combination of CNN and RNN models with an attention mechanism that is trained end-to-end. The authors show that the proposed method achieves high recognition accuracy for Korean sign language gestures and outperforms existing methods.

CHAPTER 3

SYSTEM DESIGN

3.1 Convolutional Neural Network (CNN)

Neural networks, as its name suggests, is a machine learning technique which is modeled after the brain structure. It comprises of a network of learning units called neurons. These neurons learn how to convert **input signals (**e.g. picture of a cat) into corresponding **output signals** (e.g. the label "cat"), forming the basis of automated recognition.

A convolutional neural network (CNN, or ConvNet) is a type of feed-forward artificial neural network in which the connectivity pattern between its neurons is inspired by the organization of the animal visual cortex.

CNNs have repetitive blocks of neurons that are applied across space (for images) or time (for audio signals etc). For images, these blocks of neurons can be interpreted as 2D convolutional kernels, repeatedly applied over each patch of the image. For speech, they can be seen as the 1D convolutional kernels applied across time-windows. At training time, the weights for these repeated blocks are 'shared', i.e. the weight gradients learned over various image patches are averaged.

3.1.1 CNN Summarized in 4 Steps

There are four main steps in CNN: convolution, subsampling, activation and full connectedness.

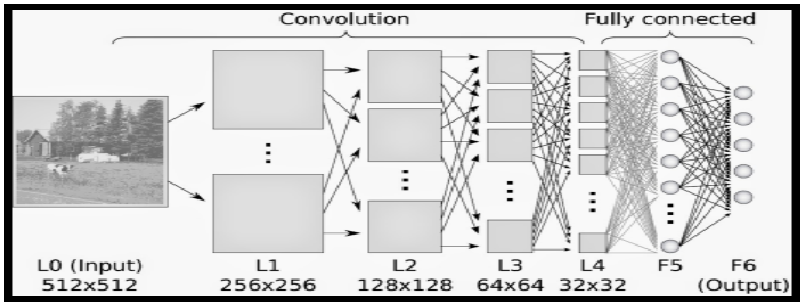

Figure 2: Convolutional neural network

3.1.1.1 Convolution

The first layers that receive an input signal are called convolution filters. Convolution is a process where the network tries to label the input signal by referring to what it has learned in the past. If the input signal looks like previous cat images it has seen before, the "cat" reference signal will be mixed into, or convolved with, the input signal. The resulting output signal is then passed on to the next layer.

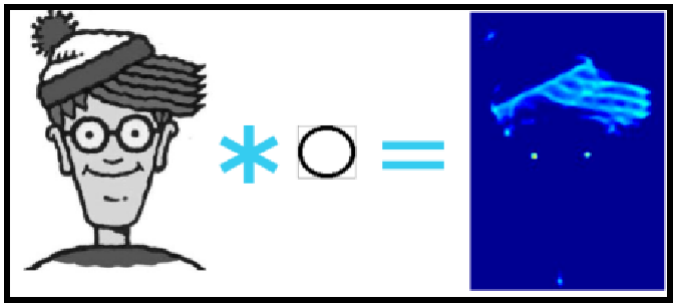

Figure 3: Convolving Wally with a circle filter. The circle filter responds strongly to the eyes.

Convolution has the nice property of being translational invariant. Intuitively, this means that each convolution filter represents a feature of interest (e.g whiskers, fur), and the CNN algorithm learns which features comprise the resulting reference (i.e. cat). The output signal strength is not dependent on where the features are located, but simply whether the features are present. Hence, a cat could be sitting in different positions, and the CNN algorithm would still be able to recognize it.

For e.g suppose we convolve a 32x32x3 (32x32 image with 3 channels R,G and B respectively) with a 5x5x3 filter.

We take the 5*5*3 filter and slide it over the complete image and along the way take the dot product between the filter and chunks of the input image.

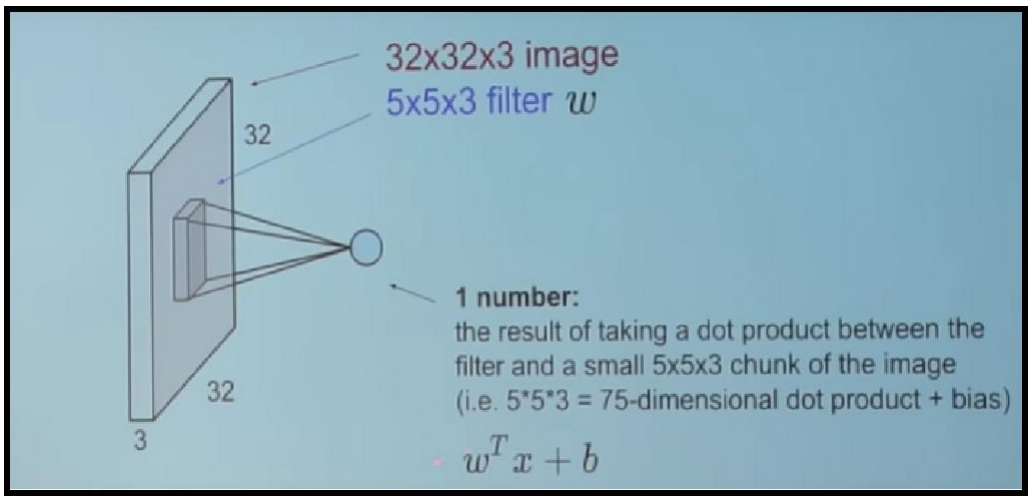

Figure 4: Dot Product of Filter with single chunk of Input Image

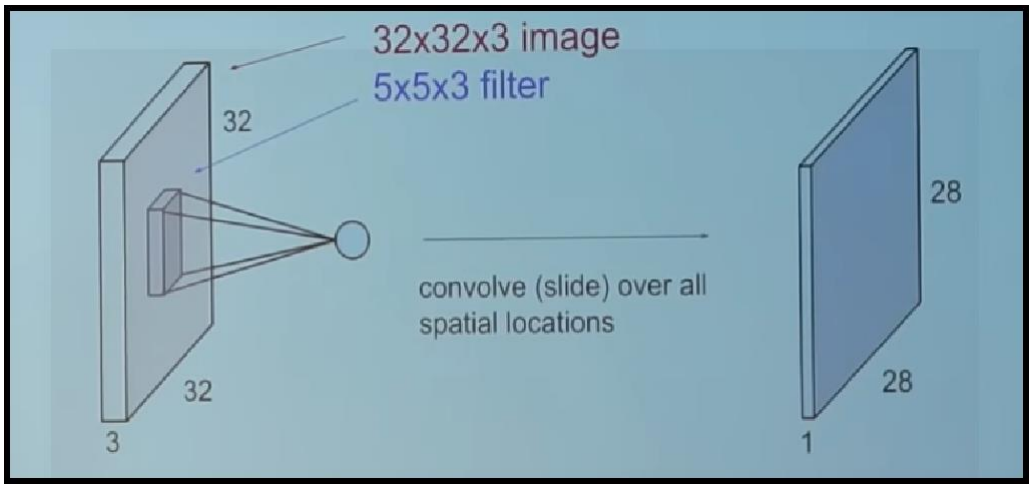

Figure 5: Dot Product or Convolve over all possible 5x5 spatial location in Input Image

The convolution layer is the main building block of a convolutional neural network. The convolution layer comprises of a set of independent filters (6 in the example shown).

Each filter is independently convolved with the image and we end up with 6 feature maps of shape 28*28*1.

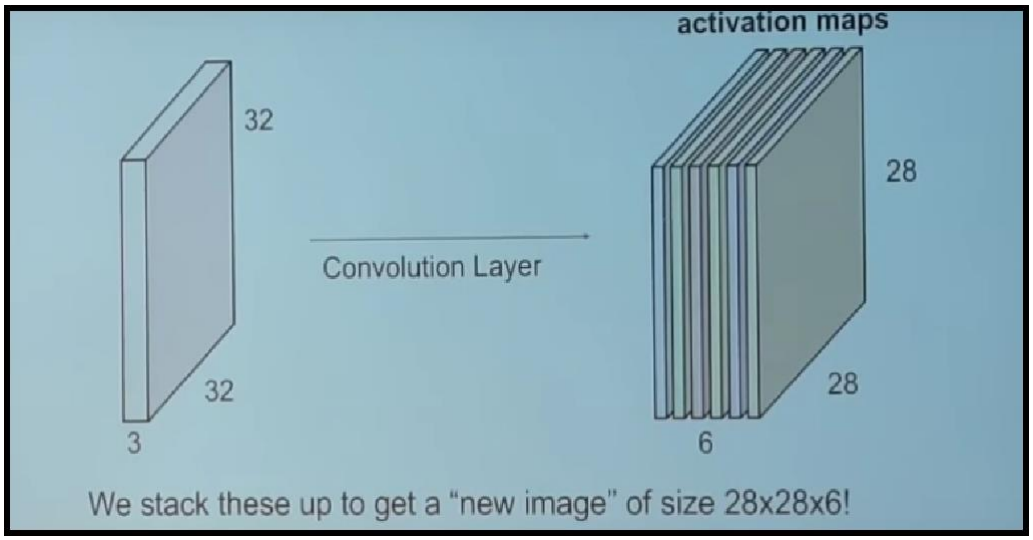

Figure 6: Input Image Convolving with a Convolutional layer of 6 independent filters

The CNN may consists of several Convolutional layers each of which can have similar or different number of independent filters. For example the following diagram shows the effect of two Convolutional layers having 6 and 10 filters respectively.

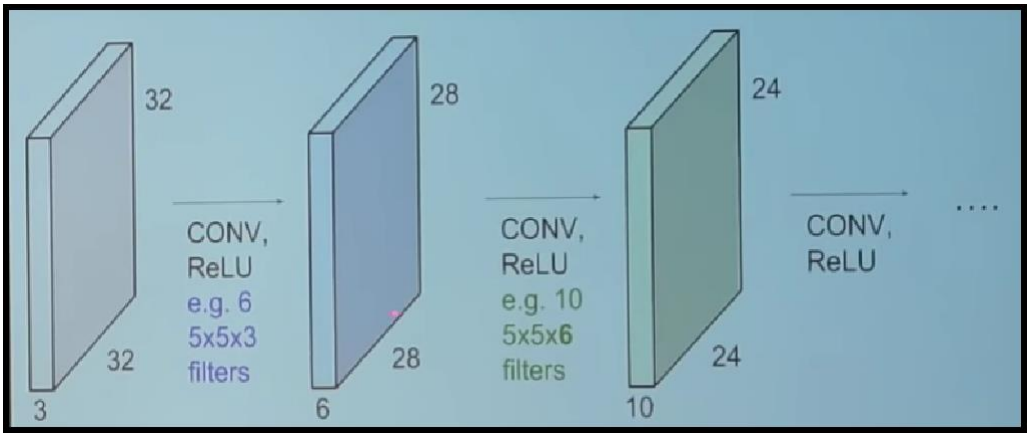

Figure 7: Input Image Convolving with two Convolutional layers having 6 and 10 filters respectively

All these filters are initialized randomly and become our parameters which will be learned by the network subsequently.

3.1.1.2 Subsampling

Inputs from the convolution layer can be "smoothened" to reduce the sensitivity of the filters to noise and variations. This smoothing process is called subsampling, and can be achieved by taking averages or taking the maximum over a sample of the signal. Examples of subsampling methods (for image signals) include reducing the size of the image, or reducing the colour contrast across red, green, blue (RGB) channels.

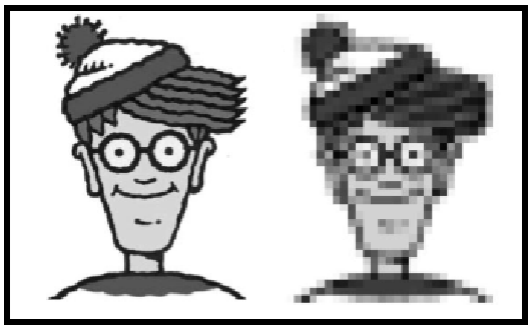

Figure 8: Sub sampling Wally by 10 times. This creates a lower resolution image.

3.1.1.3 Pooling

A pooling layer is another building block of a CNN.

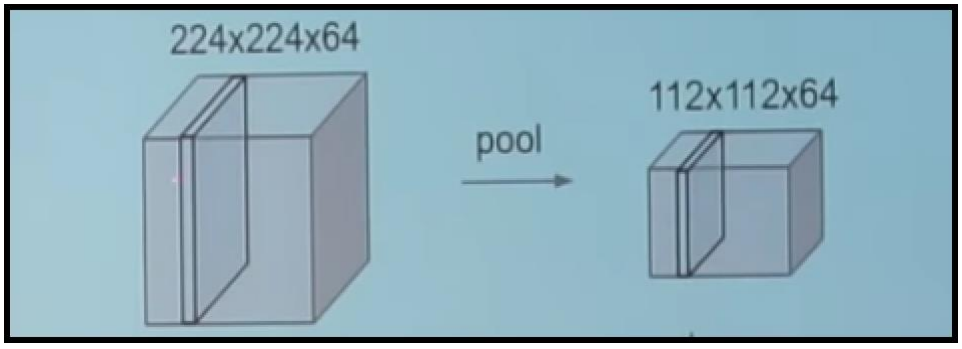

Figure 9: Pooling to reduce size from 224x224 to 112x112

Its function is to progressively reduce the spatial size of the representation to reduce the amount of parameters and computation in the network. Pooling layer operates on each feature map independently. The most common approach used in pooling is max pooling in which maximum of a region taken as its representative. For example in the following diagram a 2x2 region is replaced by the maximum value in it.

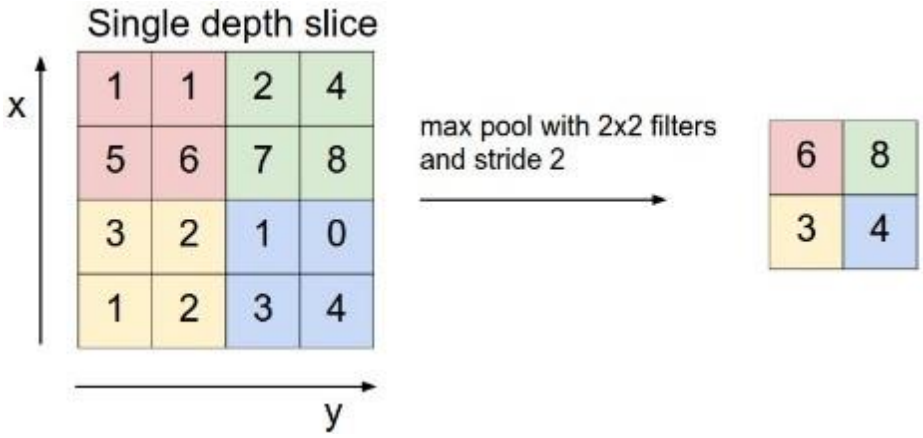

Figure 10: Max Pooling

3.1.1.4 Activation

The activation layer controls how the signal flows from one layer to the next, emulating how neurons are fired in our brain. Output signals which are strongly associated with past references would activate more neurons, enabling signals to be propagated more efficiently for identification. CNN is compatible with a wide variety of complex activation functions to model signal propagation, the most common function being the Rectified Linear Unit (ReLU), which is favoured for its faster training speed.

3.1.1.5 Fully Connected

The last layers in the network are fully connected, meaning that neurons of preceding layers are connected to every neuron in subsequent layers. This mimics

high level reasoning where all possible pathways from the input to output are considered.

3.1.1.6 (During Training) Loss

When training the neural network, there is additional layer called the loss layer. This layer provides feedback to the neural network on whether it identified inputs correctly, and if not, how far off its guesses were. This helps to guide the neural network to reinforce the right concepts as it trains. This is always the last layer during training.

3.1.2 Implementation

The algorithms used to train Convolutional Neural Networks (CNNs) can be compared to studying for an exam using flashcards. The process involves drawing flashcards with concepts, discarding the ones you know well, and revisiting the ones you're unsure of until you have a good understanding of the concepts. This method is formalized as gradient descent algorithms for forward pass learning, and modern deep learning algorithms use stochastic gradient descent, which draws the concepts at random to minimize bias. Feedback is provided through a validation set, where predictions are compared to true labels, and errors are used to refine the weights learned through backpropagation of errors. Implementing CNNs from scratch is complex, so machine learning practitioners use toolboxes such as Caffe, Torch, MatConvNet, and Tensorflow for their work.

3.1.3 Inception

We've used the Inception v3 model of the Tensor Flow library. Inception is a huge image classification model with millions of parameters that can differentiate a large number of kinds of images. We only trained the final layer of that network, so training will end in a reasonable amount of time. Inception-v3 is trained for

the ImageNet Large Visual Recognition Challenge using the data from 2012 where it reached a top-5 error rate of as low as 3.46%.

We performed transfer learning on Inception model that is we downloaded the pre-trained Inception v3 model (trained on ImageNet Dataset consisting of 1000 classes) , added a new final layer corresponding to the number of categories and then trained the final layer on the dataset.

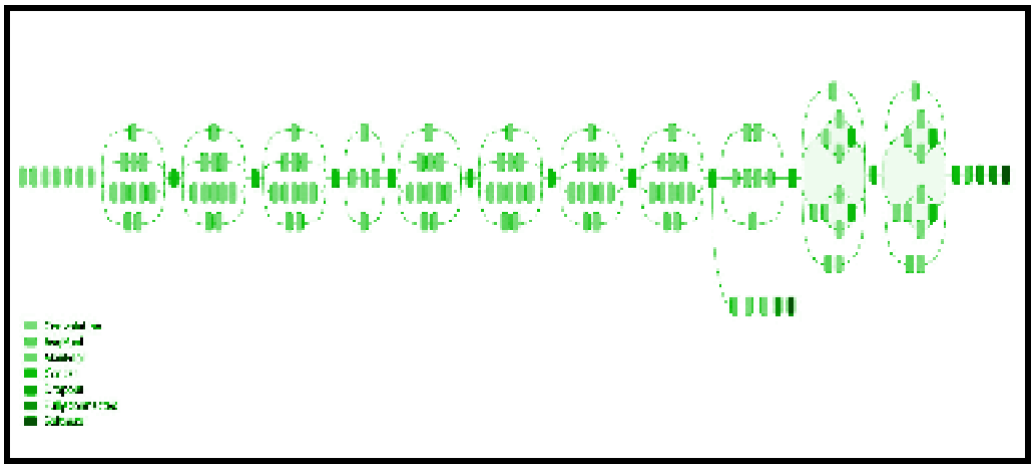

Figure 11: Inception v3 model Architecture

The kinds of information that make it possible for the model to differentiate among 1,000 classes are also useful for distinguishing other objects. By using this pre-trained network, we are using that information as input to the final classification layer that distinguishes our dataset.

3.2 Recurrent Neural Network (RNN)

Humans don't start their thinking from scratch every second. We don't throw everything away and start thinking from scratch again. Our thoughts have persistence. Traditional neural networks can't do this but Recurrent Neural Networks can. There is information in the sequence itself, and recurrent nets use it to perform tasks that feedforward networks can't. Recurrent networks are

distinguished from feedforward networks by the fact that they have feedback loop, ingesting their own outputs moment after moment as input. They're especially useful with sequential data because each neuron or unit can use its internal memory to maintain information about the previous input.

For example, in case of a network that is supposed to classify what kind of event is happening at every point in a movie. It requires the network to use its reasoning about previous events in the film to inform later ones. Another example in case of language, "I had washed my house" is much more different than "I had my house washed". This allows the network to gain a deeper understanding of the statement. This is important to note because reading through a sentence even as a human, you're picking up the context of each word from the words before it.

3.2.1 Recurrent Neural Networks have Loops

A RNN has loops in them that allow information to be carried across neurons while reading in input. In the following diagram, a chunk of Recurrent neural network, A, looks at some input xt and outputs a value ht . The loop allows information to be passed from one step of the network to the next. The decision a recurrent net reached at time step t-1 affects the decision it will reach one moment later at time step t. So recurrent networks have two sources of input, the present and the recent past, which combine to determine how they respond to new data.

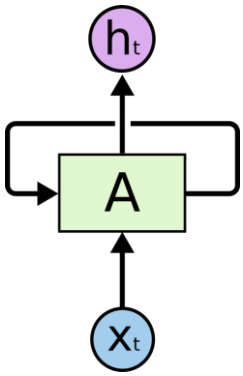

Figure 12: A chunk of Recurrent Neural Network

A recurrent neural network can be thought of as multiple copies of the same network, each passing a message to a successor. Consider what happens if we unroll the loop:

$$\mathbf{h}_t = \phi\left(W\mathbf{x}_t + U\mathbf{h}_{t-1}\right),$$

Figure 13: An Unrolled recurrent neural network

3.2.2 How Memory of previous inputs Carried forward

The hidden state at time step t is ht . It is a function of the input at the same time step xt , modified by a weight matrix W, added to the hidden state of the previous time step h_t-1 multiplied by its own hidden-state-to-hidden-state matrix U called transition matrix. The weight matrices are filters that determine how much importance to accord to both the present input and the past hidden state. The error they generate can be used to adjust their weights using Backpropagation through Time (BPTT). The sum of the weight input and hidden state function or tanh.

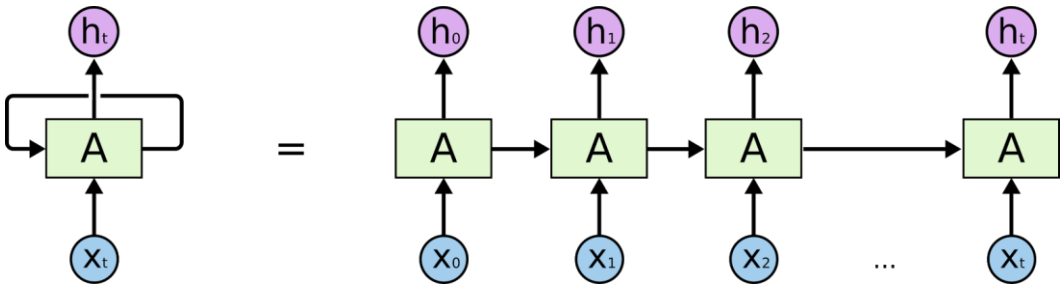

Figure 14: Memory of previous inputs being carried forward

Because this feedback loop occurs at every time step in the series, each hidden state contains traces not only of the previous hidden state, but also of all those that preceded h_t-1 for as long as memory can persist.

3.2.3 Exploding and Vanishing Gradient Problem

In theory, RNNs are absolutely capable of handling long-term dependencies. Sadly, in practice, RNNs don't seem to be able to learn them as explained. The gradient expresses the change in all weights with regard to the change in error. Since the layers and time steps of deep neural networks relate to each other through multiplication, gradient is susceptible to vanishing or exploding.

3.2.3.1 Vanishing Gradient

The gradients of the network's output with respect to the parameters in the early layers become extremely small. In other words even a large change in the value of parameters for the early layers doesn't have a big effect on the output. Hence the network can't learn the parameter effectively.

This happens because the activation functions (sigmoid or tanh) squash their input into a very small output range in a very nonlinear fashion. For example, sigmoid maps the real number line onto a "small" range of [0, 1]. As a result, there are large regions of the input space which are mapped to an extremely small range. In these regions of the input space, even a large change in the input will produce a small change in the output - hence the gradient is small.

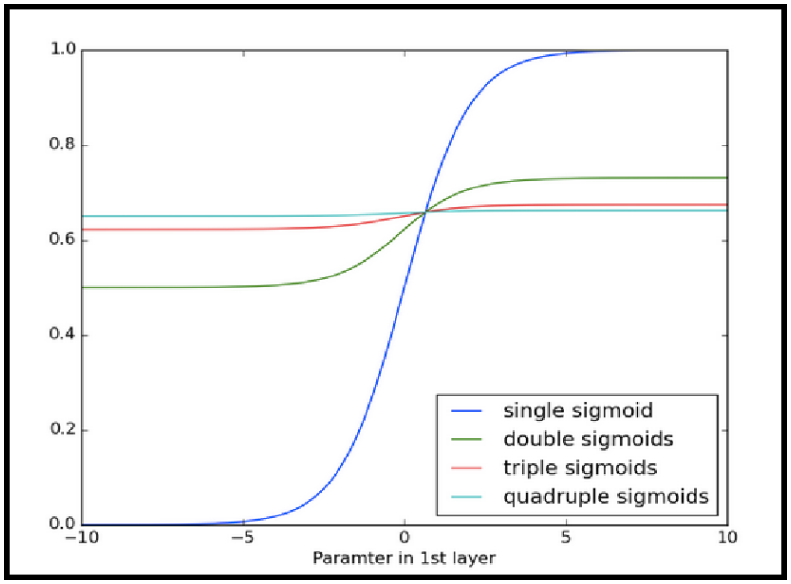

Figure 15: Vanishing Sigmoid

This becomes much worse when we stack multiple layers of such non-linearities on top of each other. For instance, first layer will map a large input region to a smaller output region, which will be mapped to an even smaller region by the second layer, which will be mapped to an even smaller region by the third layer and so on. As a result, even a large change in the parameters of the first layer doesn't change the output much. In the Fig _ we can see the effects of applying a sigmoid function over and over again. The data is flattened until, for large stretches, it has no detectable slope. This is analogous to a gradient vanishing as it passes through many layers.

3.2.3.2 Exploding Gradient

Exploding gradients treat every weight as though it were the proverbial butterfly whose flapping wings cause a distant hurricane. Those weights' gradients become saturated on the high end; i.e. they are presumed to be too powerful.

Exploding gradients can be solved relatively easily, because they can be truncated or squashed. Vanishing gradients can become too small for computers to work with or for networks to learn – a harder problem to solve.

3.2.4 Long Short-Term Memory Units (LSTMs)

A variation of recurrent net with "Long Short-Term Memory Units" LSTMs, was proposed by the German researchers Sepp Hochreiter and Juergen Schmidhuber as a solution to the vanishing gradient problem. LSTMs help preserve the error that can be backpropagated through time and layers. By maintaining a more constant error, they allow recurrent nets to continue to learn over many time steps (over 1000), thereby opening a channel to link causes and effects remotely. LSTMs are explicitly designed to avoid the long-term dependency problem. Remembering information for long periods of time is practically their default behaviour, not something they struggle to learn!

3.2.5 Our RNN Model

We have create a RNN model based on LSTMs. The first layer is an input layer used to feed input to the upcoming layers. Its size is determined by the size of the input being fed. Our Model is a wide network consisting of single layer of 256 LSTM units. This Layer is followed by a fully connected layer with SoftMax activation. In Fully Connected every neuron is connected to every neuron of previous layer. Finally a regression layer to apply a regression (linear or logistic) to the provided input. We used Adam(Adaptive Moment Estimation) which is a stochastic optimizer, as a gradient descent optimizer to minimize the provided loss function "categorical_crossentropy" (which calculate the errors)

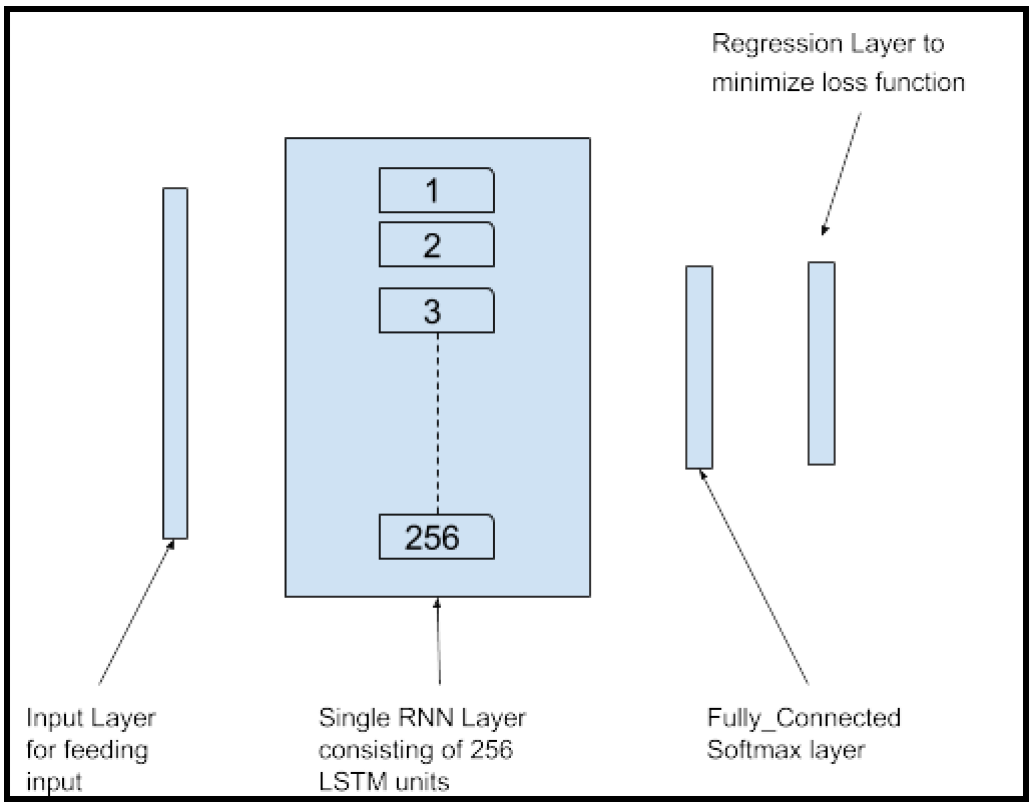

Figure 16: Our RNN Model

We also tried a wider RNN network with 512 LSTM units and another deep RNN network with three layers of 64 LSTM units each. We tested these on a sample of the dataset and found that wide model with 256 LSTM units performed the best and therefore only the wide model was used for training and testing on complete dataset.

CHAPTER 4

SYSTEM ARCHITECTURE

In this chapter, the System Architecture for Video-Based Sign Language Gesture Accuracy Prediction using CNN-RNN Model is represented and the modules are explained.

4.1 SYSTEM ARCHITECTURE DIAGRAM

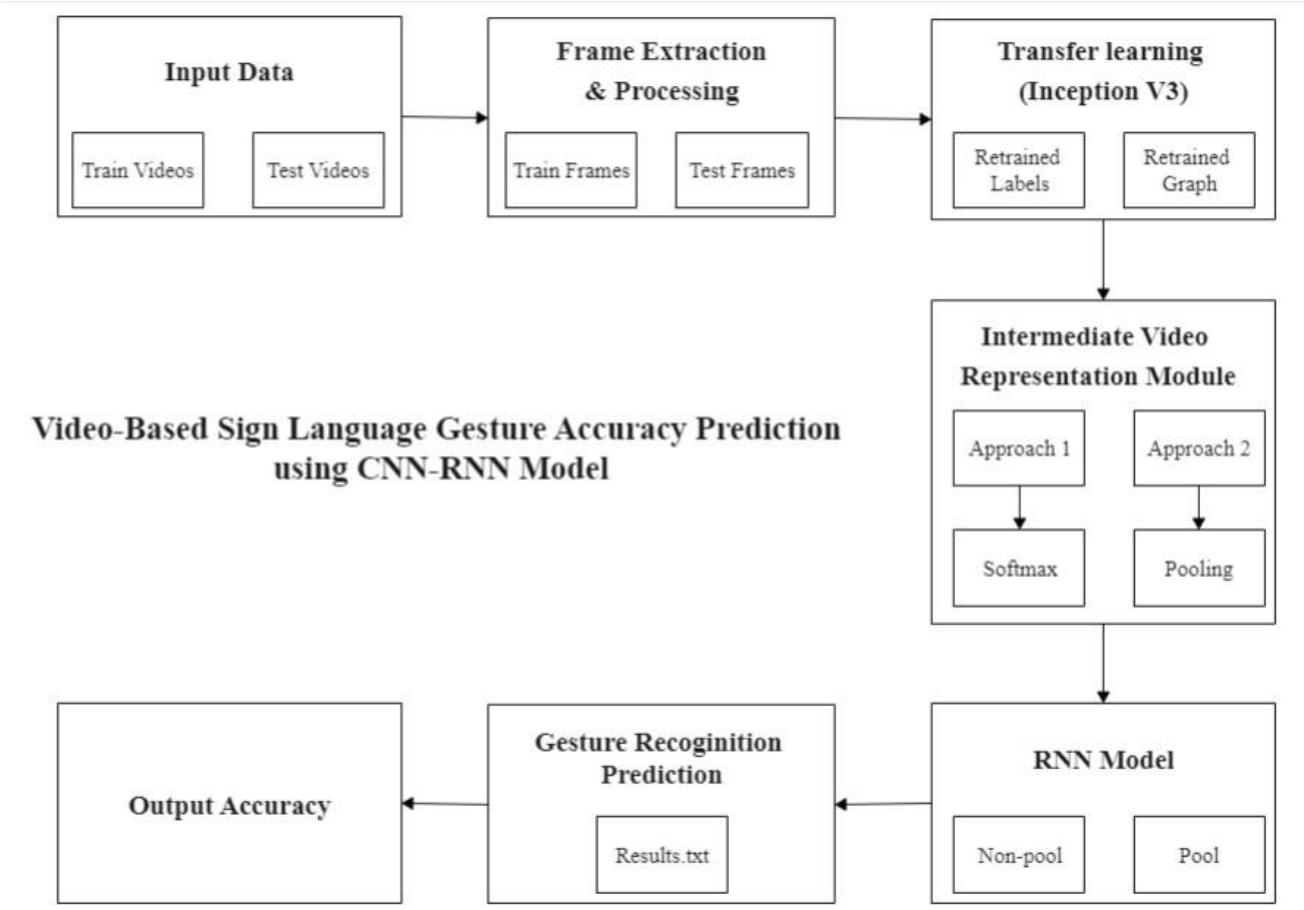

Figure 17: System Architecture Diagram

4.2 ARCHITECTURE DESCRIPTION

System architecture of our project provides the detailed description about the datasets, pre-processing of data, extraction of data, training of data using CNN and RNN algorithms. This process involves building a machine learning model that can recognize different hand gestures in videos.

The data set used in this project consists of Argentinian Sign Language (LSA) gestures, which includes approximately 200 videos belonging to 4 gesture categories. LSA is the sign language used by the deaf community in Argentina. This data set is unique because it focuses specifically on LSA, which is a language that is not as widely studied as some other sign languages.

The first step is to prepare the data by creating two folders, one for training videos and another for testing videos. Within each folder, create subfolders corresponding to each gesture category and fill them with videos containing those gestures.

Next, extract frames from each video in both the training and testing sets. Then, use transfer learning to retrain the Inception v3 image classification model on the extracted frames to recognize the hand gestures. This will create two files, retrained_labels.txt and retrained_graph.pb.

There are two approaches to represent the videos in an intermediate format. The first approach involves representing each video as a sequence of n dimensional vectors (output of SoftMax) where n is the number of gesture categories. The second approach involves representing each video as a sequence of 2048-dimensional vectors (output of last pool layer). These representations will be stored in separate files for the training and testing data.

CHAPTER 5

SYSTEM IMPLEMENTATION

We have used two approaches to train the model on the temporal and the spatial features. Both approaches differ by the inputs given to RNN to train it on the temporal features

5.1 Data Set Used

The data set used for both the approaches consists of Argentinian Sign Language(LSA) Gestures, with around 200 videos belonging to 4 gestures categories. 10 non-expert subjects executed the 5 repetitions of every gesture thereby producing 50 videos per category or gesture.

ID	Name
1	Opaque
2	Red
3	Yellow
4	Green

Out of the 50 gestures per category, 75% i.e. 40 were used for training and 25% i.e. 10 were used for testing

5.2 First Approach

In this approach we extracted spatial features for individual frames using inception model (CNN) and temporal features using RNN. Each video (a sequence of frames) was then represented by a sequence of predictions made by CNN for each of the individual frames. This sequence of predictions was given as input to the RNN.

5.2.1 Methodology

- First, we will extract the frames from the multiple video sequences of each gesture.
- After the first step, noise from the frames i.e background, body parts other than hand are removed to extract more relevant features from the frame.
- Frames of the train data are given to the CNN model for training on the spatial features. We have used inception model for this purpose which is a deep neural net.
- Store the train and test frame predictions. We'll use the model obtained in the above step for the prediction of frames.
- The predictions of the train data are now given to the RNN model for training on the temporal features. We have used LSTM model for this purpose.

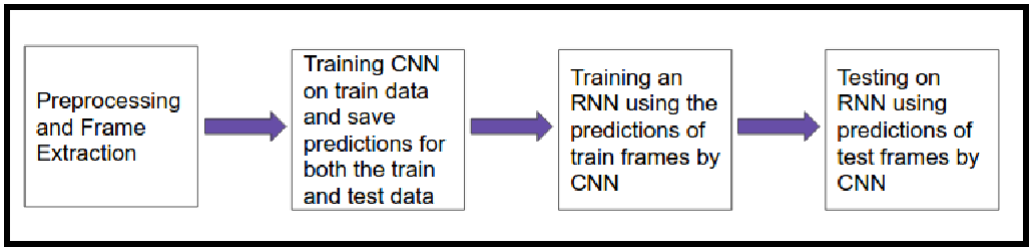

Figure 18: Approach 1 Methodology

In further subsections of this section, each step of the methodology has been shown diagrammatically for better understanding of that step.

5.2.1.1 Frame Extraction and Background Removal

Each video gesture video is broken down into a sequence of frames. Frames are then processed to remove all the noise from the image that is everything except hands. The final image consists of grey scale image of hands to avoid color specific learning of the model

Figure 19: One of the Extracted Frames

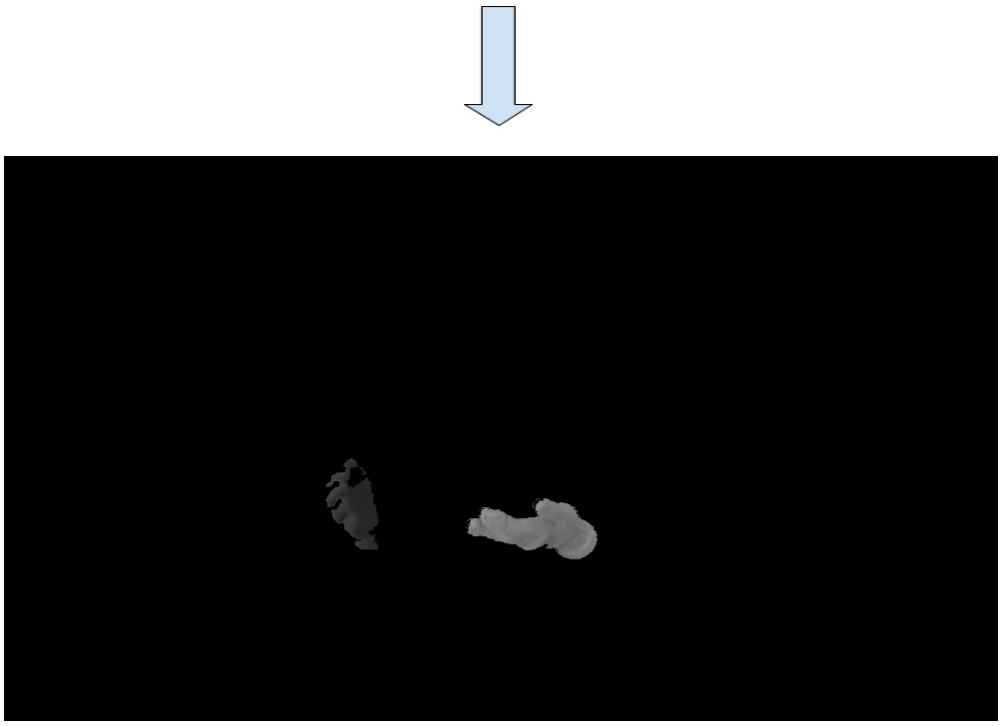

Figure 20: Frame after extracting hands (Background Removal)

5.2.1.2 Train CNN(Spatial Features) and Prediction

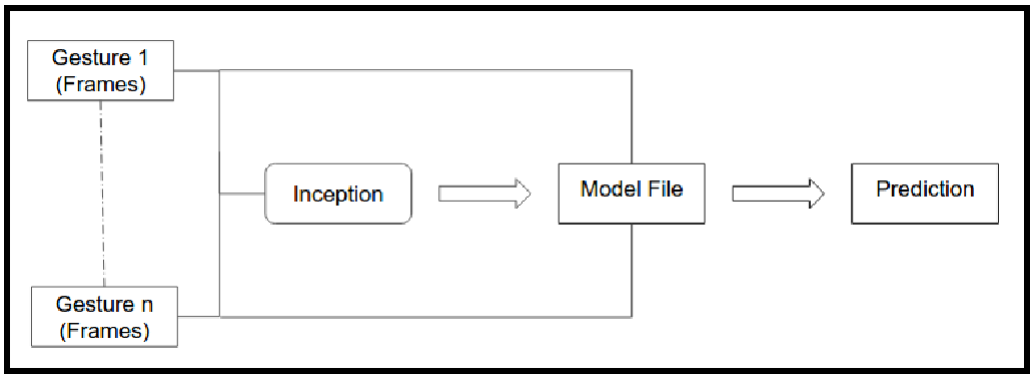

Figure 21: Train CNN and Prediction

The first row in the below illustration is the video of a gesture Elephant. The second row shows the set of frames extracted from it. The third row shows the sequence of predictions for each frame by CNN after training it.

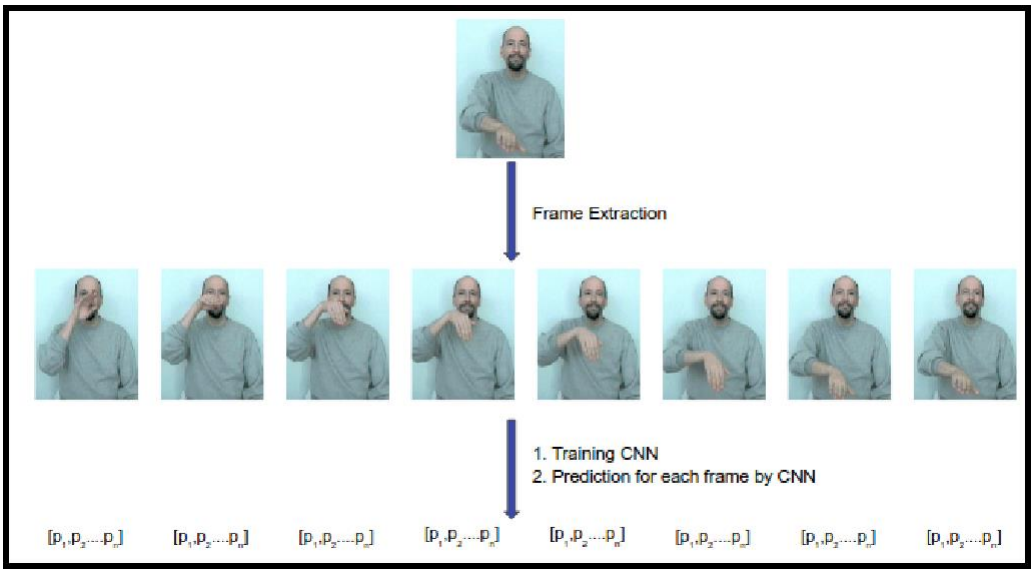

Figure 22: Train CNN and Prediction Illustration

5.2.1.3 Training RNN (Temporal Features)

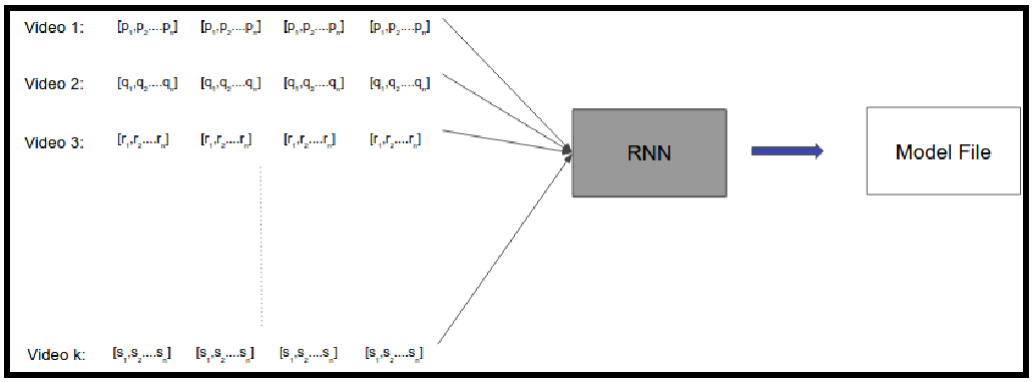

Figure 23: Train RNN Illustration

5.2.2 Limitations

The CNN makes probabilistic predictions for each frame in the video sequence, and the length of these predictions corresponds to the number of classes to be classified (in this case, 4 classes). This means that the feature vector for each frame in the RNN is also dependent on the number of classes, with fewer classes resulting in shorter feature vectors.

5.3 Second Approach

In this approach we have used CNN to train the model on the spatial features and have given the output of the pool layer, before it's made into a prediction, to the RNN. The pool layer gives us a 2048 dimensional vector that represents the convoluted features of the image, but not a class prediction.

Rest of the steps of this approach are same as that of first approach. Both approaches only differ by inputs given to RNN.

CHAPTER 6

RESULTS AND CODING

6.1 SAMPLE CODE

```
//Utilities used by our other RNN scripts.
from collections import deque
from sklearn.model_selection import train_test_split
from tflearn.data_utils import to_categorical
import tflearn
import numpy as np
import pickle
def get_data(input_data_dump, num_frames_per_video, labels, ifTrain):
    """Get the data from our saved predictions or pooled features."""
    # Local vars.
    X = []
    y = []
    temp_list = deque()
    # Open and get the features.
    with open(input_data_dump, 'rb') as fin:
        frames = pickle.load(fin)
        for i, frame in enumerate(frames):
```

```
        features = frame[0]
        actual = frame[1].lower(
        # frameCount = frame[2]
        # Convert our labels into binary.
        actual = labels[actual
        # Add to the queue.
        if len(temp_list) == num_frames_per_video - 1:
            temp_list.append(features)
            flat = list(temp_list)
            X.append(np.array(flat))
            y.append(actual)
            temp_list.clear()
        else:
            temp_list.append(features)
            continue
print("Class Name\tNumeric Label")
for key in labels:
    print("%s\t\t%d" % (key, labels[key]))

# Numpy.
X = np.array(X)
y = np.array(y)
```

```
    print("Dataset shape: ", X.shape)
    # One-hot encoded categoricals.
    y = to_categorical(y, len(labels))
    # Split into train and test.
    if ifTrain:
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
        return X_train, X_test, y_train, y_test
    else:
        return X, y
def get_network(frames, input_size, num_classes):
    """Create our LSTM"""
    net = tflearn.input_data(shape=[None, frames, input_size])
    net = tflearn.lstm(net, 128, dropout=0.8, return_seq=True)
    net = tflearn.lstm(net, 128)
    net = tflearn.fully_connected(net, num_classes, activation='softmax')
    net = tflearn.regression(net, optimizer='adam',
                             loss='categorical_crossentropy', name="output1")
    return net

def get_network_deep(frames, input_size, num_classes):
```

```
    """Create a deeper LSTM"""
    net = tflearn.input_data(shape=[None, frames, input_size])
    net = tflearn.lstm(net, 64, dropout=0.2, return_seq=True)
    net = tflearn.lstm(net, 64, dropout=0.2, return_seq=True)
    net = tflearn.lstm(net, 64, dropout=0.2)
    net = tflearn.fully_connected(net, num_classes, activation='softmax')
    net = tflearn.regression(net, optimizer='adam',
                             loss='categorical_crossentropy', name="output1")
    return net


def get_network_wide(frames, input_size, num_classes):
    """Create a wider LSTM"""
    net = tflearn.input_data(shape=[None, frames, input_size])
    net = tflearn.lstm(net, 256, dropout=0.2)
    net = tflearn.fully_connected(net, num_classes, activation='softmax')
    net = tflearn.regression(net, optimizer='adam',
                             loss='categorical_crossentropy', name='output1')
    return net


def get_network_wider(frames, input_size, num_classes):
    """Create a wider LSTM"""
    net = tflearn.input_data(shape=[None, frames, input_size])
```

```
    net = tflearn.lstm(net, 512, dropout=0.2)
    net = tflearn.fully_connected(net, num_classes, activation='softmax')
    net = tflearn.regression(net, optimizer='adam',
                             loss='categorical_crossentropy', name='output1')
return net
//Given a saved output of predictions or pooled features from our CNN,
train an RNN (LSTM) to examine temporal dependencies
from rnn_utils import get_network_wide, get_data
import argparse
import tensorflow as tf
import tflearn
import os
def load_labels(label_file):
    label = {}
    count = 0
    proto_as_ascii_lines = tf.gfile.GFile(label_file).readlines()
    for l in proto_as_ascii_lines:
        label[l.strip()] = count
        count += 1
    return label
def main(input_data_dump, num_frames_per_video, batch_size, labels,
model_file):
```

```
# Get our data.
    X_train, X_test, y_train, y_test = get_data(input_data_dump, num_frames_per_video, labels, True)
    num_classes = len(labels)
    size_of_each_frame = X_train.shape[2]

    # Get our network.
    net = get_network_wide(num_frames_per_video, size_of_each_frame, num_classes

    # Train the model.
    try:
        model = tflearn.DNN(net, tensorboard_verbose=0)
        model.load('checkpoints/' + model_file)
        print("\nModel already exists! Loading it")
        print("Model Loaded")
    except Exception:
        model = tflearn.DNN(net, tensorboard_verbose=0)
        print("\nNo previous checkpoints of %s exist" % (model_file))
    model.fit(X_train, y_train, validation_set=(X_test, y_test),
              show_metric=True, batch_size=batch_size, snapshot_step=100,
              n_epoch=10)
```

```
# Save it.
    x = input("Do you wanna save the model and overwrite? y or n: ")
    if(x.strip().lower() == "y"):
        model.save('checkpoints/' + model_file)


if __name__ == '__main__':
    parser = argparse.ArgumentParser(description='Train a RNN')
    parser.add_argument("input_file_dump", help="file containing intermediate representation of gestures from inception model")
    parser.add_argument("model_file", help="Name of the model file to be dumped. Model file is created inside a checkpoints folder")
    parser.add_argument("--label_file", help="path to label file generated by inception", default="retrained_labels.txt")
    parser.add_argument("--batch_size", help="batch Size", default=32)
    args = parser.parse_args()
    labels = load_labels(args.label_file)
    input_data_dump = args.input_file_dump
    num_frames_per_video = 201
    batch_size = args.batch_size
    model_file = args.model_file
    main(input_data_dump, num_frames_per_video, batch_size, labels, model_file)
```

6.2 Result of Approach 1

```
Model Exists! Loading it
Model Loaded
{0: 'green', 1: 'opaque', 2: 'red', 3: 'yellow'}
Accuracy:  100.0
```

Figure 24: Result of Approach 1

Average accuracy obtained using this approach is 100%.

6.3 Result of Approach 2

Out of the 40 Gestures (10 Per category) used for testing recognized correctly giving an average accuracy of 100%.

Category Wise Accuracy is tabulated and is given on the next page.

ID	Gesture	Accuracy
1	Opaque	100
2	Red	100
3	Yellow	100
4	Green	100

Figure 25: Accuracy

The second approach also provided a better accuracy as first approach because of the fact that in the first approach the input to the RNN was a sequence of 46 dimensional prediction while in the second approach the RNN was being given a sequence of 2048 dimensional pool layer output. This gave RNN more number of feature points to distinguish among different videos.

6.4 Performance Analysis

```
No previous checkpoints of non_pool.model exist
---------------------------------
Run id: KFBBF9
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 160
Validation samples: 40
--
Training Step: 5  | time: 16.685s
| Adam | epoch: 001 | loss: 0.00000 - acc: 0.7132 | val_loss: 1.26864 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 10  | time: 2.609s
| Adam | epoch: 002 | loss: 0.00000 - acc: 0.5314 | val_loss: 0.94912 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 15  | time: 2.515s
| Adam | epoch: 003 | loss: 0.00000 - acc: 0.7654 | val_loss: 0.63082 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 20  | time: 2.635s
| Adam | epoch: 004 | loss: 0.00000 - acc: 0.8407 | val_loss: 0.37580 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 25  | time: 2.587s
| Adam | epoch: 005 | loss: 0.00000 - acc: 0.8924 | val_loss: 0.22395 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 30  | time: 2.548s
| Adam | epoch: 006 | loss: 0.00000 - acc: 0.8663 | val_loss: 0.15308 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 35  | time: 2.564s
| Adam | epoch: 007 | loss: 0.00000 - acc: 0.8224 | val_loss: 0.12030 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 40  | time: 2.628s
| Adam | epoch: 008 | loss: 0.00000 - acc: 0.7879 | val_loss: 0.10649 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 45  | time: 2.577s
| Adam | epoch: 009 | loss: 0.00000 - acc: 0.9153 | val_loss: 0.10098 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 50  | time: 2.542s
| Adam | epoch: 010 | loss: 0.00000 - acc: 0.8998 | val_loss: 0.10163 - val_acc: 1.0000 -- iter: 160/160
--
Do you wanna save the model and overwrite? y or n: y
```

Figure 26: Approach 1

```
No previous checkpoints of pool.model exist
---------------------------------
Run id: KCBO3O
Log directory: /tmp/tflearn_logs/
---------------------------------
Training samples: 160
Validation samples: 40
--
Training Step: 5  | time: 36.200s
| Adam | epoch: 001 | loss: 0.00000 - acc: 0.3947 | val_loss: 1.26453 - val_acc: 0.4000 -- iter: 160/160
--
Training Step: 10  | time: 14.411s
| Adam | epoch: 002 | loss: 0.00000 - acc: 0.4573 | val_loss: 0.95129 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 15  | time: 13.988s
| Adam | epoch: 003 | loss: 0.00000 - acc: 0.8610 | val_loss: 0.65459 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 20  | time: 14.073s
| Adam | epoch: 004 | loss: 0.00000 - acc: 0.8741 | val_loss: 0.43863 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 25  | time: 13.834s
| Adam | epoch: 005 | loss: 0.00000 - acc: 0.8768 | val_loss: 0.33529 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 30  | time: 13.593s
| Adam | epoch: 006 | loss: 0.00000 - acc: 0.8539 | val_loss: 0.30627 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 35  | time: 13.868s
| Adam | epoch: 007 | loss: 0.00000 - acc: 0.7947 | val_loss: 0.26689 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 40  | time: 14.774s
| Adam | epoch: 008 | loss: 0.00000 - acc: 0.8332 | val_loss: 0.18850 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 45  | time: 14.049s
| Adam | epoch: 009 | loss: 0.00000 - acc: 0.9266 | val_loss: 0.14114 - val_acc: 1.0000 -- iter: 160/160
--
Training Step: 50  | time: 13.690s
| Adam | epoch: 010 | loss: 0.00000 - acc: 0.9055 | val_loss: 0.13511 - val_acc: 1.0000 -- iter: 160/160
--
Do you wanna save the model and overwrite? y or n: y
```

Figure 27: Approach 2

6.5 Graph Analysis

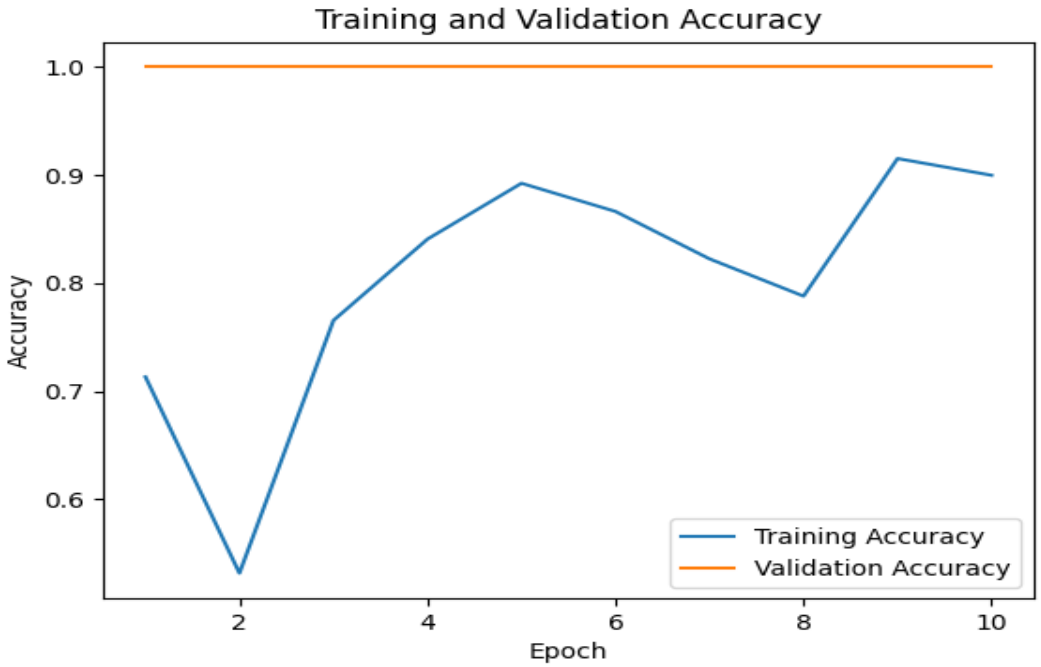

Figure 28: Approach 1 Accuracy

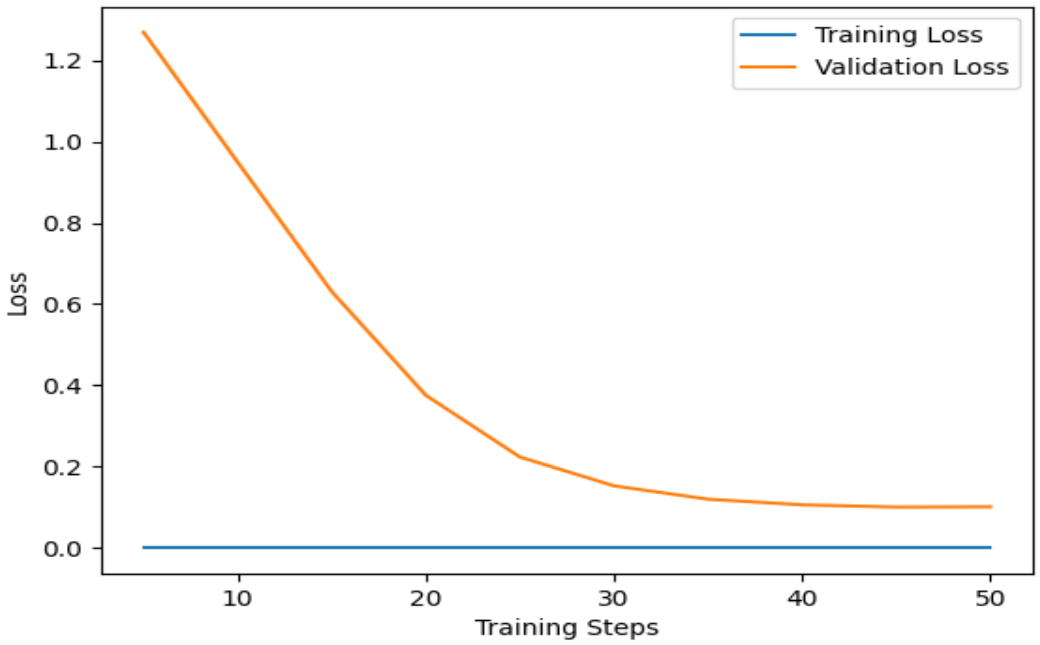

Figure 29: Approach 2 Loss

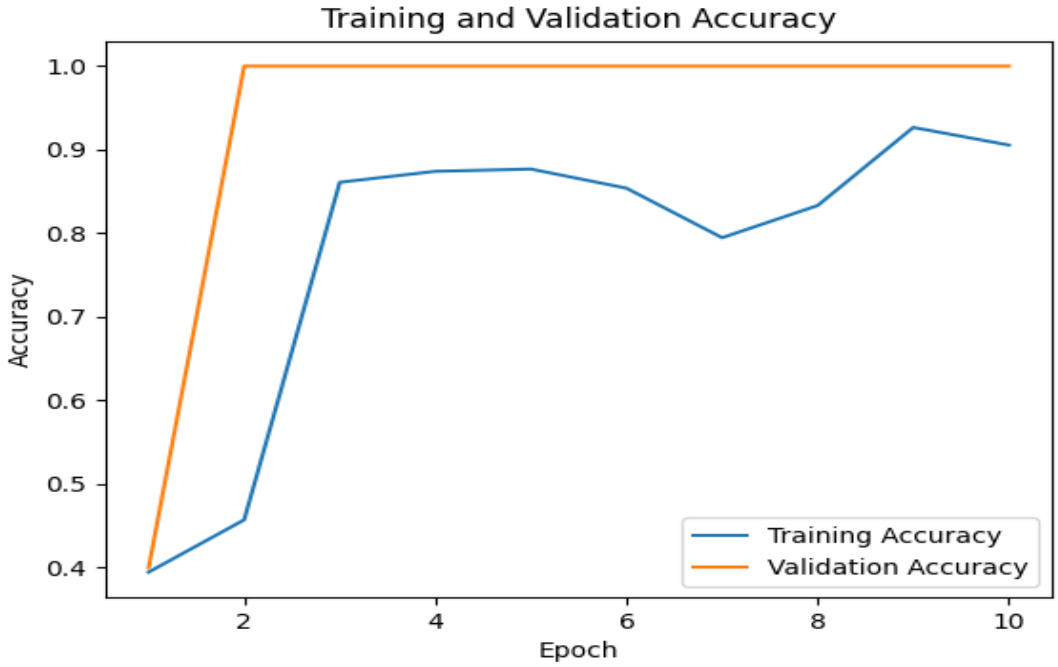

Figure 30: Approach 2 Accuracy

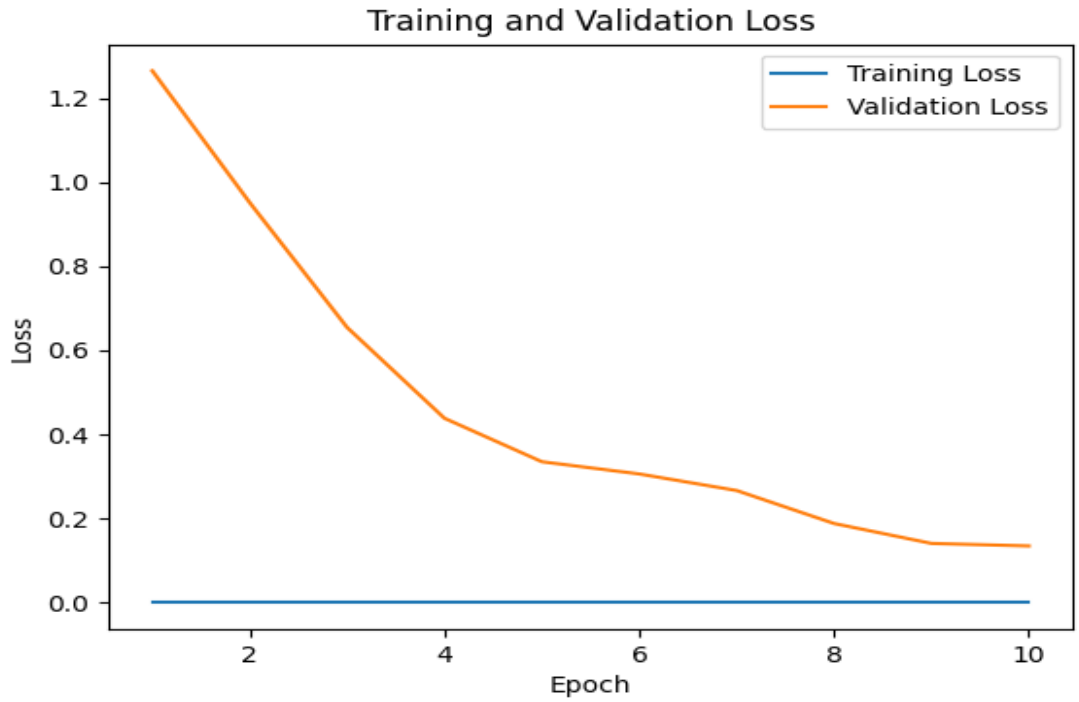

Figure 31: Approach 2 Loss

CHAPTER 7

CONCLUSION AND FUTURE WORK

Hand gestures are a powerful way for human communication, with lots of potential applications in the area of human computer interaction. Vision based hand gesture recognition techniques have many proven advantages compared with traditional devices. However, hand gesture recognition is a difficult problem and the current work is only a small contribution towards achieving the results needed in the field of sign language gesture recognition. This report presented a system able to interpret isolated hand gestures from the Argentinian Sign Language (LSA).

Videos are difficult to classify because they contain both the temporal as well as the spatial features. We have used two different models to classify on the spatial and temporal features. CNN was used to classify on the spatial features whereas RNN was used to classify on the temporal features. We obtained an accuracy of 100 %. This shows that CNN along with RNN can be successfully used to learn spatial and temporal features and classify Sign Language Gestures.

We have used two approaches to solve our problem and both of the approaches only differ by the inputs given to the RNN as explained in the methodologies above.

We wish to extend our work further in recognising continuous sign language gestures with real time gesture recognition. This method for individual gestures can also be extended for sentence level sign language. Also the current process uses two different models, training inception (CNN) followed by training RNN. For future work one can focus on combining the two models into a single model.

REFERENCES

[1] H. Alabkari, H. Elgibreen, and M. R. Al-khaldi, "Sign Language Recognition Based on Convolutional Neural Networks," in Proceedings of the 2019 International Conference on Computing, Networking and Communications (ICNC), 2019.

[2] M. Al-Rousan, M. Al-Fahoum, and A. Al-Rababah, "Arabic Sign Language Recognition System Using Deep Neural Networks," in Proceedings of the 2020 International Conference on Advanced Machine Learning Technologies and Applications (AMLTA), 2020.

[3] N. Farhadi and M. Tabrizi, "Indian Sign Language Recognition Using Convolutional Neural Networks," in Proceedings of the 2019 4th International Conference on Robotics and Automation Engineering (ICRAE), 2019

[4] A. Hamad, H. Aly, and F. E. Abd El-Samie, "Real-Time Hand Gesture Recognition for Arabic Sign Language Using Convolutional Neural Network," in Proceedings of the 2019 IEEE 9th Annual Computing and Communication Workshop and Conference (CCWC), 2019

[5] S. K. Hossain, M. H. Rahman, and M. R. Islam, "Bangla Sign Language Recognition Using a Combination of Convolutional and Recurrent Neural Networks with Spatial and Temporal Attention Mechanisms," Sensors, vol. 20, no. 20, 2020.

[6] S. G. Kim, S. H. Kim, and K. I. Chang, "Sign Language Recognition using CNN with Temporal Convolutional Layers," in Proceedings of the 2020 International Conference on Electronics, Information, and Communication (ICEIC), 2020.

[7] J. Liang, K. Lai, and Y. Zhang, "Sign Language Recognition Based on Hybrid CNN-RNN Model," Journal of Physics: Conference Series, vol. 1496, no.1, 2020.

[8] Nandy, Anup, Jay Shankar Prasad, Soumik Mondal, Pavan Chakraborty, and Gora Chand Nandi. "Recognition of isolated indian sign language gesture in real time." Information Processing and Management (2010): 102¬107.

[9] Ronchetti, Franco, Facundo Quiroga, César Armando Estrebou, and Laura Cristina Lanzarini. "Handshape recognition for argentinian sign language using probsom." Journal of Computer Science & Technology 16 (2016).

[10] Singha, Joyeeta, and Karen Das. "Automatic Indian Sign Language Recognition for Continuous Video Sequence." ADBU Journal of Engineering Technology 2, no. 1 (2015).

[11] Tripathi, Kumud, and Neha Baranwal GC Nandi. "Continuous Indian Sign Language Gesture Recognition and Sentence Formation." Procedia Computer Science 54 (2015): 523-531.

[12] L. Wu and X. Zhu, "Deep Learning for Gesture Recognition in American Sign Language," in Proceedings of the 2018 IEEE International Conference on Computer Vision (ICCV), 2018.

[13] Y. Xiang, Y. Huang, and X. Li, "Chinese Sign Language Recognition Using Convolutional and Recurrent Neural Networks," in Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), 2017.

[14] J. Ye, W. Hu, and Q. Liu, "Sign Language Recognition Using Convolutional Neural Network and Recurrent Neural Network with Attention Mechanism," in Proceedings of the 2019 IEEE 3rd Information Technology, Networking, Electronic and Automation Control Conference (ITNEC), 2019.

[15] H. Yoo, K. Lee, and K. Park, "End-to-End Sign Language Recognition with CNN and RNN," in Proceedings of the 2021 IEEE International Conference on Consumer Electronics (ICCE), 2021.