Spring 2-18-2019

American Sign Language Recognition Using Machine Learning and Computer Vision

Kshitij Bantupalli

Ying Xie
College of Computing and Software Engineering - Information Technology

AMERICAN SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING AND COMPUTER VISION

A Thesis Presented to

Dr Selena He

Faculty of College of Computing and Software Engineering

By

Kshitij Bantupalli

In Partial Fulfillment

Of Requirements for the Degree

Master of Science – Computer Science

Kennesaw State University

December 2018

AMERICAN SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING AND COMPUTER VISION

Approved:

_____

Dr. Selena He

_____

Dr Dan Lo

_____

John Preston

In presenting this thesis as a partial fulfillment of the requirements for an advanced degree from Kennesaw State University, I agree that the university library shall make it available for inspection and circulation in accordance with its regulations governing materials of this type. I agree that permission to copy from, or to publish, this thesis may be granted by the professor under whose direction it was written, or, in his absence, by the dean of the appropriate school when such copying or publication is solely for scholarly purposes and does not involve potential financial gain. It is understood that any copying from or publication of this thesis which involves potential financial gain will not be allowed without written permission.

_____

Kshitij Bantupalli

Notice to Borrowers

Unpublished theses deposited in the Library of Kennesaw State University must be used only in accordance with the stipulations prescribed by the author in the preceding statement.

The author of this thesis is:

Kshitij Bantupalli

3840 Fenway Crossing, Marietta, GA 30075

The director of this thesis is:

Dr. Selena He

363 J Building, 1100 South Marietta Pkwy

Marietta, GA 30060

Users of this thesis not regularly enrolled as students at Kennesaw State University are requirement to attest acceptance of the preceding stipulations by signing below. Libraries borrowing this thesis for use of their patrons are required to see that each user records here the information requested.

Name of user	Address	Date	Type of use (Examination only or copying)

AMERICAN SIGN LANGUAGE RECOGNITION USING MACHINE LEARNING AND COMPUTER VISION

A Thesis Presented to

Dr Selena He

Faculty of College of Computing and Software Engineering

By

Kshitij Bantupalli

In Partial Fulfillment

Of Requirements for the Degree

Master of Science – Computer Science

Kennesaw State University

December 2018

Abstract

Speech impairment is a disability which affects an individual's ability to communicate using speech and hearing. People who are affected by this use other media of communication such as sign language. Although sign language is ubiquitous in recent times, there remains a challenge for non-sign language speakers to communicate with sign language speakers or signers. With recent advances in deep learning and computer vision there has been promising progress in the fields of motion and gesture recognition using deep learning and computer vision-based techniques. The focus of this work is to create a vision-based application which offers sign language translation to text thus aiding communication between signers and non-signers. The proposed model takes video sequences and extracts temporal and spatial features from them. We then use Inception, a CNN (Convolutional Neural Network) for recognizing spatial features. We then use an RNN (Recurrent Neural Network) to train on temporal features. The dataset used is the American Sign Language Dataset.

Table of Contents

List of Figures

List of Tables

Chapter I

Introduction

Sign language is a form of communication used by people with impaired hearing and speech. People use sign language gestures as a means of non-verbal communication to express their thoughts and emotions. But non-signers find it extremely difficult to understand, hence trained sign language interpreters are needed during medical and legal appointments, educational and training sessions. Over the past five years, there has been an increasing demand for interpreting services. Other means, such as video remote human interpreting using high-speed internet connections, have been introduced. They will thus provide an easy to use sign language interpreting service, which can be used, but has major limitation such as accessibility to internet and an appropriate device.

To address this, we use an ensemble of two models to recognize gestures in sign language. We use a custom recorded American Sign Language dataset based off an existing dataset [1] for training the model to recognize gestures. The dataset is very comprehensive and has 150 different gestures performed multiple times giving us variation in context and video conditions. For simplicity, the videos are recording at a common frame rate. We propose to use a CNN (Convolutional Neural Network) named Inception to extract spatial features from the video stream for Sign Language Recognition (SLR). Then by using a LSTM (Long Short-Term Memory), an RNN (Recurrent Neural Network) model, we can extract temporal features from the video sequences.

A proposed improvement is to test the model with more gestures to see how accuracy scales with larger sample sizes and compare the performance of two different outputs of a CNN. Another proposed improvement is to use newer technologies and compare performance to see if the model can have better performance.

Chapter II

Literature Review

Literature review the problem shows that there have been several approaches to address the issue of gesture recognition in video using several different methods. One of the messages used Hidden Markov Models (HMM) to recognize facial expressions from video sequences combined with Bayesian Network Classifiers and Gaussian Tree Augmented Naive Bayes Classifier. Francois also published a paper on human posture recognition in a video sequence using methods based on 2 D and 3 D appearance. The work mentions using PCA to recognize silhouettes from a static camera and then using 3 D to model posture for recognition. This approach has the drawback of having intermediary gestures which may lead to ambiguity in training and therefore lower accuracy in prediction.

Let's approach the analysis of video segments using neural networks which involves extracting visual information in the form of feature vectors. Neural networks do face issues such as tracking of hands, segmentation of subject from the background and environment, illumination, variation, occlusion, movement and position. The paper splits the dataset into segments, extracts features and classifies using Euclidean distance and K-nearest neighbor.

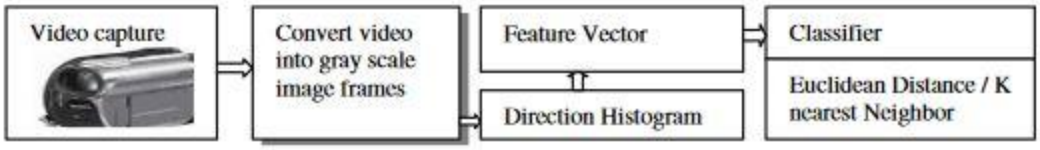


Figure 1 Workflow of Nandy et al.

Work done by blank defines how to do continuous Indian sign language recognition. The paper proposes frame extraction from video data, preprocessing the data, extracting key frames from the data followed by extracting other features, recognition and finally optimization. Preprocessing is done by converting the video to a sequence of RGB frames. Each frame having the same dimensions. Skin color segmentation is used to extract skin regions with the help of AHS we gradient. The images of obtained were converted to binary form. Food keyframes were extracted by calculating a gradient between the frames. And features were extracted from the keyframes using an orientation histogram. Classification was done by Euclidean distance, Manhattan distance, chess board distance and Mahalanobis distance.

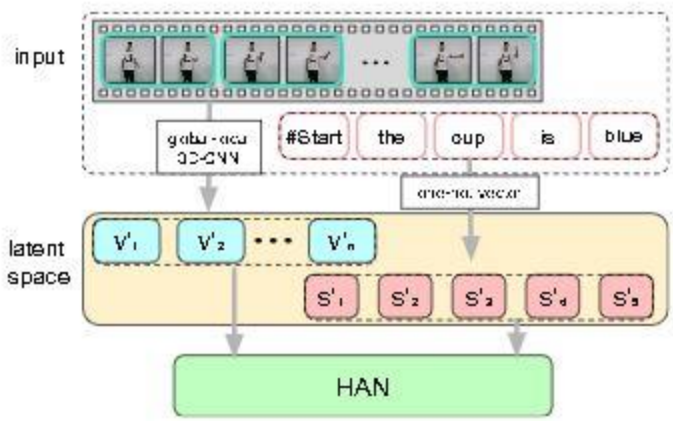


Figure 2 LS-HAN Network

In a paper by Jie et al. [2], the authors recognized problems in SLR such as problems in recognition when the signs are broken down to individual words and the issues with continuous SLR. They decided to solve the problem without isolating individual signs, which removes an extra level of preprocessing (temporal segmentation) and another extra layer of post-processing because they believed that temporal segmentation is crucial to SLR and without its errors propagate into subsequent steps. Combined with the strenuous labelling of individual words adds a huge challenge to SLR without temporal segmentation. They addressed this issue with a new framework called Hierarchical Attention Network with Latent Space (LS-HAN), which eliminates the preprocessing of temporal segmentation. The framework consists of a two-stream CNN for video feature representation generation, a Latent Space for semantic gap bridging and a Hierarchical Attention Network for space-based recognition.

Other approaches to SLR include using an external device such as a Leap Motion controller to recognize movement and gestures such as the work done by Chong *et al.* [3]. The study differs from other work because it includes the complete grammar of the American Sign Language which consists of 26 letters and 10 digits. The work is aimed dynamic movements and extracting features to study and classify them. The experimental results have been promising with accuracies of 80.30% for Support Vector Machines (SVM) and 93.81% for Deep Neural Networks (DNN).

Research in the fields of hand gesture recognition also aid to SLR research such as the work by Linqin *et al.* [4] . In it, the authors have used RGB-D data to recognize human gestures for human-computer interaction. They approach the problem by calculating Euclidean distance between hand joints and shoulder features to generate a unifying feature descriptor. A dynamic time warping (IDTW) algorithm is proposed to obtain final recognition results, which works by applying weighted distance and restricted search path to avoid major computation costs unlike conventional approaches. The experimental results of this method show an average accuracy of 96.5% and better. The idea is to develop real time gesture recognition which could also be extended to SLR.

The work done by Ronchetti *et al.* [5] on the Argentinian sign language offers another approach to the problem; using a database of handshapes of the Argentinian Sign Language and a technique for processing images, extracting descriptors and handshape classification using ProbSom. The technique is very similar to Support Vector Machines (SVM),

Random Forests and Neural Networks. The overall accuracy of the approach was upwards of 90%.

Hardie *et al.* [6] used an external device called Myo armband to collect data about the position of a user's hands and fingers over time. The authors of the paper use these technologies along with sign language translation as they consider each sign a combination of gestures. We use the same approach of considering each sign as a gesture. The paper utilizes a dataset collected by a group at University of South Wales, which contains parameters, such as hand positions, hand rotation, and finger bend for 95 unique signs. Each sign has an input stream and they predict which sign the stream falls into. The classification is made using SVM and logistic regression models. Lower quality of the data requires a more sophisticated approach, so they explore different methods of temporal classification.

The literature review shows that there have been different approaches to this problem within neural networks itself. The input feed to the neural networks plays a big role in how the architecture of the network is shaped, such is a 3DCNN model would take RGB input along with the depth field. So, for the purpose of validation the results of our model were compared to two very similar approaches to the problem. Lu *et al.* [7] used a general CNN network to extract spatial features and used an LSTM to extract sequence features. Vivek *et al.* [8] used CNN models with RGB inputs for their architecture. The authors of [8] worked on American Sign Language with a custom dataset of their own making. The architecture in [7] was a pretrained CNN called ResNet along with a custom LSTM of their

design whereas [8] used a CNN for stationary hand gestures so we had to take the liberty of extending their base model with the LSTM from our network.

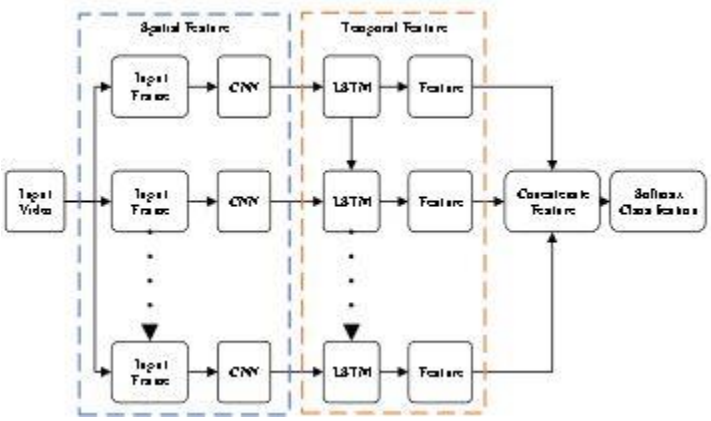
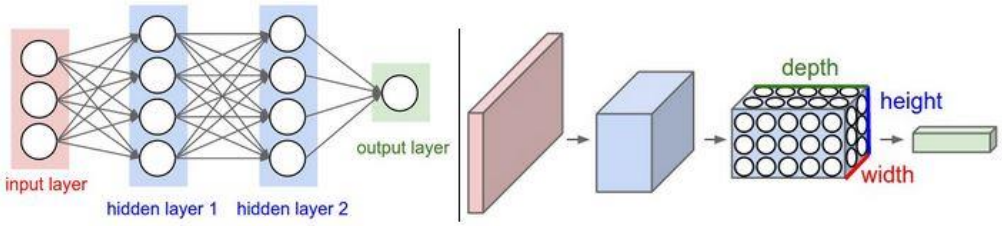


Figure 3 Model for Lu et al.

Chapter III

Machine Learning using Convolutional Neural Networks (CNN)

CNN's or ConvNets are a category of neural networks which are respectable in the field of image recognition and classification. [9]CNN's use multilayer perceptron's which require minimal preprocessing to "train" the architecture to perform the task of recognition/classification very effective. [10]CNN's were modelled to perform like biological processes in terms of connectivity patterns between neurons in the visual cortex of animals. CNN's tend to perform better than other image and video recognition algorithms in fields of image classification, medical image analysis and natural language processing.



Left: A regular 3-layer Neural Network. Right: A ConvNet arranges its neurons in three dimensions (width, height, depth), as visualized in one of the layers. Every layer of a ConvNet transforms the 3D input volume to a 3D output volume of neuron activations. In this example, the red input layer holds the image, so its width and height would be the dimensions of the image, and the depth would be 3 (Red, Green, Blue channels).

Figure 4 A ConvNet Architecture Explained

LeNet Architecture

LeNet was one of very first CNN's to be mainstream and paved the way for future research into the field of multilayer perceptron's and CNN's alike. This revolutionary work by Yann LeCun [9] was named LeNet5 and was a result of many previous successful iterations since 1988. The LeNet was developed mainly for character recognition tasks such as digits and zip codes. Since then the MNIST dataset [11] was created and is still curated as a benchmark to test every new proposed neural network architecture for accuracy.

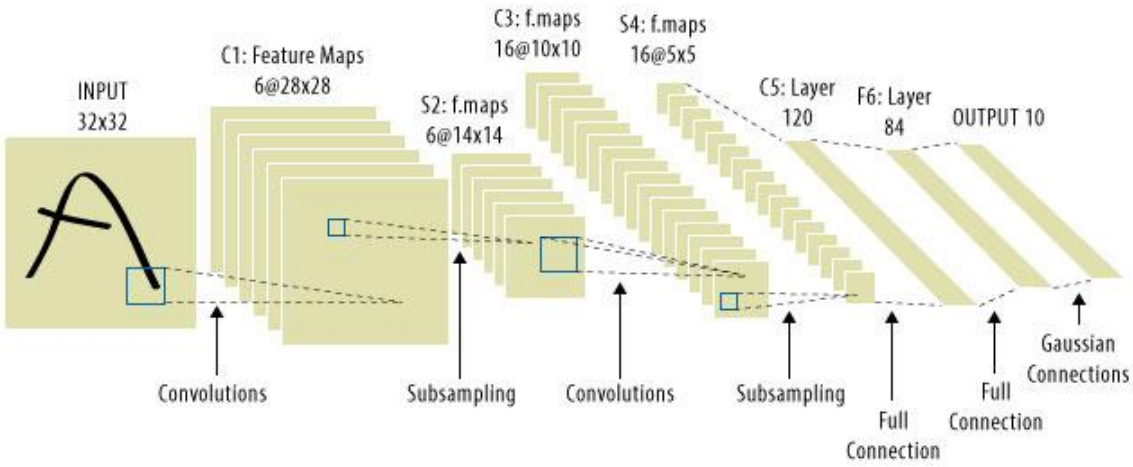


Figure 5 A Simple ConvNet

The CNN in the figure above takes a 32x32 image from the MNIST handwritten image dataset and classifies it to a possible of 26 letters in the English alphabet.

There are four key operations in the image above:

1.Convolution

2.Non-Linearity (ReLU)

3. Pooling or Sub Sampling

4. Classification (Fully Connected Layer)

These operations are the fundamental blocks of *every* CNN. Let's try to understand each in detail and how each operation works.

Convolution

Let's take the example of MNIST handwritten digits. If we represent each image as a matrix of pixel values we have a 2 D matrix with values ranging from 0 to 255. ConvNets derive the word convolution from this step. The purpose of convolution is to extract features from the input data whether it be image, video or sequential data. Convolution works by preserving spatial relationships between pixels by learning image features using small ROI's. If we go back to considering each image as a matrix of pixels, convolution summarizes blocks of data from the matrix to a smaller dimension.

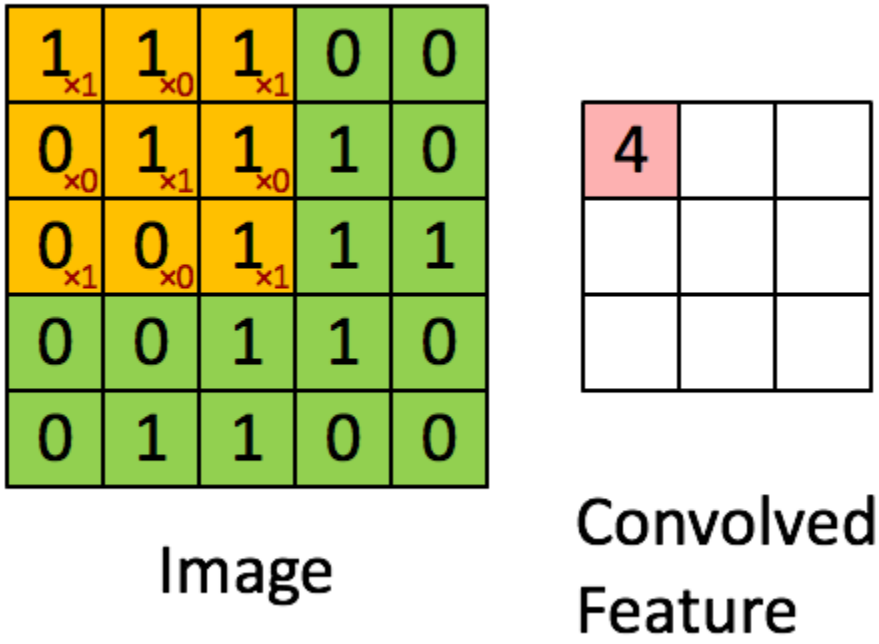


Figure 6 Convolution Operation and Convolved Feature

The computation is achieved by computing element wise multiplication and adding the outputs to get a representation of the ROI over the original image.

In CNN terminology, the 3x3 matrix is called a “filter” or “feature detector” and the matrix formed by sliding the filter over the image is called a “Convolved Feature” or “Feature Map”. The process is repeated till the input image is converted to a series of feature maps.

There are several options the convolution function can use to generate a feature map such as:

1. Edge Detection
2. Sharpen
3. Blur

All these are achieved just by changing the numeric values of the filter matrix before the convolution operation [12], this means that different filters achieve different results depending on what the end goal of the model is.

In theory, CNN “learns” the values of the filters during the training process thanks to backpropagation of gradients and loss between the perceptron’s of the neural network. The size of the filter, architecture and the number of filters is modifiable to achieve different results. The size of the feature map is a product of:

1. Depth: The depth of the volume determines the number of connected neurons in a layer and the input volume. We train these neurons through backpropagation to learn features. Let’s say we take an image classification

problem; the input layer takes an image and succeeding layers extract features from the image such as edges, blobs etc.

2. Stride: Stride refers to the number of pixels by which the filter moves over the input image. When the stride is 4, it moves 4 pixels after forming a feature map. Having a larger stride forms smaller feature map. Having a lower stride size leads to heavily overlapping receptive fields between columns. When resulting fields overlap less, then we get smaller spatial dimensions/feature maps [13]*)*..
3. Zero-padding: Sometimes the input matrix is padded with zeroes to apply the filter to elements in the border of the image. Adding zero padding is called wide convolution which is different from narrow convolution which is the case for non-padded images. The size of the padding can be considered a hyperparameter and it provides control of the output volume spatial size.

Connectivity is an issue to consider when dealing with high-dimensional inputs such as images, because connecting all the neurons with previous volumes does not take spatial structure into account. CNN's take advantage of local connection between neurons of nearby layers, the extent of which is a hyperparameter called receptive field. The connects are always in local in space, but they extend to the depth of input volume.

Free parameters are controlled in convolutional layers by using the concept of parameter sharing. It relies on the assumption that a patch feature is reusable and can be used in different layers of the neural network.

ReLU

The Non-Linearity operation is used after the convolution operation mentioned above. ReLU stands for Rectified Linear Unit and is a non-linear operation [14]. It is applied to each element individually and it replaces all negative pixel values in the feature map to zero. The purpose of the ReLU is to introduce non-linearity since real world training is non-linear and the CNN should model to that.

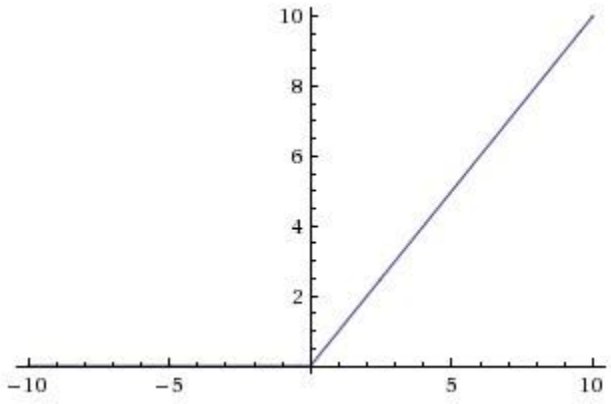


Figure 7 ReLU Operation

ReLU function works by giving a max between an input number and 0.

$$A(x) = \max(0, x)$$

ReLU is also a very computationally cheap activation function unlike other activation functions such as *sigmoid* and *tanh* because it requires simpler mathematical operations which is a good point to consider when designing neural networks.

The Pooling Step

Pooling or subsampling is a layer which reduces dimensionality of feature maps generated from the convolutional layer while retaining the most important information [15]. Pooling can be of several different types such as: Max, Average, Sum etc.

Max Pooling is when we define a window of a certain size and take the largest element from it. Instead of taking the largest element we could also take an average (Average Pooling) or sum all the elements in it (Sum Pooling). We continue to move the filter over the entire image like the stride we took in convolution till we have a pooled layer of the specified type in the architecture.

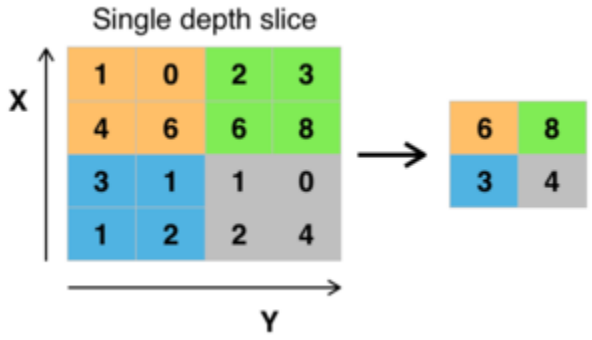


Figure 8 Max Pooling Layer

The pooling layer further reduces the dimensionality of the input image and therefore reduces the number of parameters and computations in the network. It gives us a representation of the input image in a cleaner more concise form.

Fully Connected Layer

The fully connected layer is a multilayer perceptron which uses activation functions such as SoftMax in the output layer. There are several activation functions like SoftMax, but we shall only discuss SoftMax for the purposes of thesis. The term fully connected layer implies that every neuron in the layer is connected to every neuron in the previous layer. The convolutional layer along with the pulling layer generates a summarization of the original input image which is fed into the fully connected layer. The fully connected layer send gives an output which can be either classification or regression.

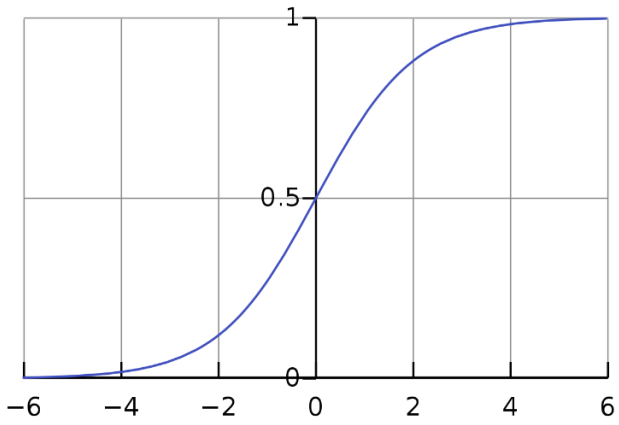


Figure 9 SoftMax Function

The fully connected layer allows for operation such as backpropagation which are key features which enable a neural network to perform classification with a high accuracy like it does. The SoftMax layer uses a SoftMax function to squash a vector between zero and one and it is the most used activation function in classification.

The Complete Architecture

Now that we have covered individual elements of a CNN, lets look at how it all comes together. The training of the CNN can be summarized as follows:

1. Initialize all filters and parameters to perform the convolution step on the input images.
2. The architecture takes an input image and goes through all the above steps in a sequential order and finds an output. The outputs are propagated backwards through the network to "train" the network.
3. Step 2 is repeated till the predicted outputs are close to ground truth and cannot be changed further.

The above-mentioned steps essentially train the neural network to perform a specific task. When a new image is introduced to the neural network after it has been trained it can predict a class of the image based on the training dataset. The training dataset plays a huge role in the accuracy and performance of the model.

Inception

Inception is a Google Net model designed to classify images with astound accuracy in 2014. It was developed by Yann LeCun and his team working at Google who came up with an innovative solution to increasing accuracy of a deep neural network instead of stacking layers. Inception was a heavily engineered, complex network which used a lot of tricks to get high speed and accuracy from the model.

Since its creation Inception v1 has gone to become Inception v2, Inception v3, Inception v4 and finally Inception ResNet. We have used Inception v4 in the ensemble of models for SLR in our work.

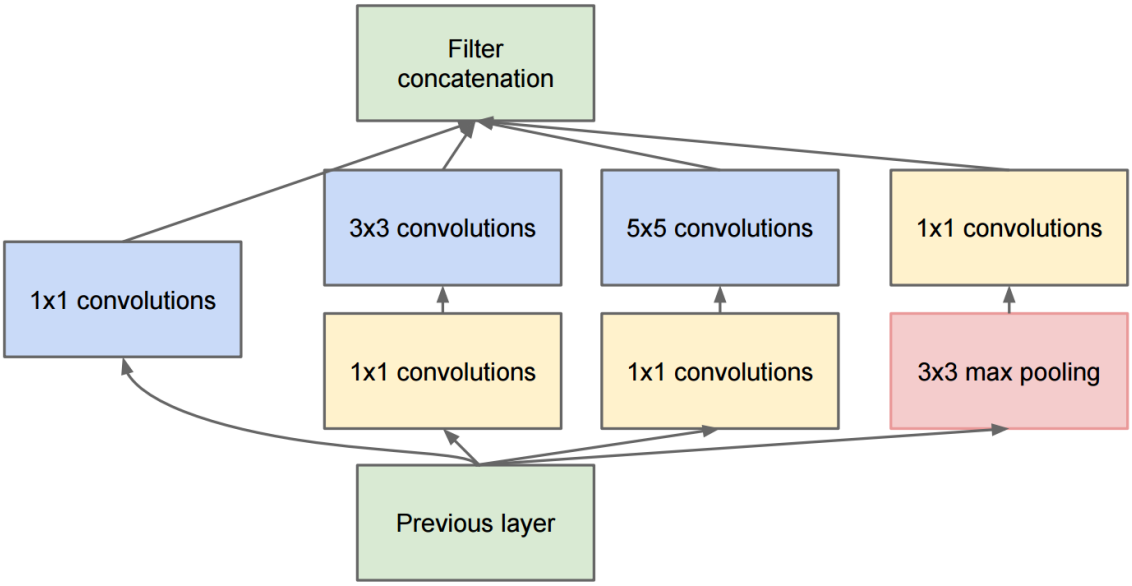


Figure 10 Inception Module, Naive Version

Let's look at the Inception module which is what sets the network apart from other models and what guarantees high levels of accuracy and performance.

Revisiting the problem of selecting appropriate filter size and type of activation function of the pooling layer when designing a neural network. The task of testing every combination with trial and error is not only time consuming but also trivial. You may/may not find the optimal combination. Inception solves this performing all possible parameters in parallel before going to the next layer. If the next layer is also an Inception module, then each of the convolutions feature maps will be passed through the mixture of convolutions of the current layer. The idea is that you don't have to spend time worrying about hyper-parameters when the network can do that for you.

Regularization Methods

Regularization is an approach to introducing additional information to solve an ill-posed problem to prevent overfitting. Let's look at the different types of regularization which CNN's offer.

1. Dropout: The fully connected layer is prone to overfitting due to the number of parameters it occupies [16]. The prevent this we use dropout. At each training stage, individual nodes are "dropped out" of the net with either probability *1-p* or kept with probability *p*. The removed nodes are then reinserted into the network with original weights. The probability that a hidden node would be dropped is 0.5 and

during testing we find an average of all possible 2^n. Dropout decreases overfitting by avoiding training all nodes on all training data. Dropout also significantly improves training speed.

2. DropConnect: DropConnect [17] is the generalization of dropout in which each connect can be dropped with probability *1-p*. DropConnect is like dropout, it introduces dynamic sparsity, but the sparsity is on the weights rather than the output vectors.

Chapter IV

Recurrent Neural Networks

Recurrent Neural Networks (RNN) are a class of neural networks where neurons are directly connected to form a directed graph. One of the features of an RNN is the ability to exhibit temporal dynamic behavior for a time sensitive sequence. RNN's are known for high accuracy in time bound sequences of data because of the structure and layout of the network [18].

There are 2 broad classes of recurrent neural networks where one is finite impulse and the other is infinite impulse. Both the classes of networks exhibit dynamic behavior [19]. The infinite impulse recurrent network is a directed cyclic graph whereas the finite impulse is directed acyclic.

Traditional RNN's operate on the assumption that outputs produced from predeceasing layers are important for successive accuracies. RNN's can retain information which makes them very effective in working with sequential data. They can retain information because of "memory" which captures information about outputs.

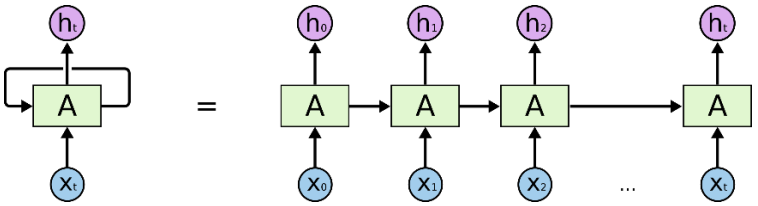


Figure 11 A Simplified Representation of an RNN

Figure 8 shows the internal working of an RNN. As we can see the outputs *h* from the previous layer is fed to the next layer along with new input *x* from the sequence. The arrows denote the hidden memory and the "storing" of previous outputs and feeding to the next layer.

The hidden state of the RNN is basically the memory of the network and it captures information from previous time steps. An RNN also uses the same parameters at all the layers unlike the CNN architecture. The frequency of the outputs from RNN can be tweaked depending on how often it is required.

What can RNN's do?

RNN's are used in natural language processing tasks quite often [20]. The most commonly used version of RNN's are called Long Short-Term Memory or LSTM's which is also what we use for our model for SLR. RNN's can perform a variety of tasks such as:

1. Language Modelling and Generating Text: RNN's can be used to predict the probability a word occurring in a sentence such as a generative model. Such models

are called Language Models and they take an input of words and give a possible combination of outputs of words.

2. Machine Translation: Machine Translation is like Language Modelling in which we output a sequence of words, the difference between the two is that Language Translation does not require completed text to generate new text whereas Machine Translation requires completed text to give an output.
3. Speech Recognition: If an RNN is given a sequence of audio signals, it can be possibly trained to recognize speech and meaningful sentences from recorded audio.

Vanishing Gradient Problem

The vanishing gradient problem is a trouble found in training artificial neural networks. In such neural networks, the weights of the neural networks are updated after every pass. But the problem is sometimes the gradient would be vanishingly small, which prevents the weight from changing values. The worst case being that the neural network stops updating entirely.

Let's take an example of a traditional activation function such as the hyperbolic tangent function which have gradients in the range (0, 1) and uses backpropagation. Over *n* layers, the gradient decreases exponentially and the front *n* layers train very slowly.

Solutions to vanishing gradient:

1. Multi-level hierarchy: Jurgen Schmidhuber proposed multi-layer hierarchy of networks which are pretrained through unsupervised learning and then fine-tuned through backpropagation.
2. Related approach: Similar approaches have been used in feed forward neural networks which are used to classify labeled data. A deep belief network model by Hinton involves learning the distribution of a high-level model using latent variables.
3. LSTM: Will be covered in the next section.
4. Faster hardware: Hardware has been advancing since neural networks were introduced to the computing world, making backpropagation feasible for several layers deeper than vanishing gradient is recognized for.
5. Residual networks: One of the newer ways to deal with vanishing gradient problem is to use residual networks called ResNet. ResNet have a higher training error than a shallow network meaning that data was disappearing through the layers of the network, and the output of the shallow layer diminishes through the layers of the deeper network yielding poorer results. We can improve the performance by splitting the deep network into chinks and passing the input into each chunk directly. ResNet yields lower training error than their shallower counterparts by simply reintroducing outputs from shallower layers.
6. Other activation functions: Rectifiers such as ReLU suffer less from vanishing gradient problem.

LSTM

Long short-term memory (LSTM) networks were discovered in 1997 and set accuracy records in multiple application domains [21].

Around 2007, LSTM's started to be used extensively for speech recognition with incredible results and soon after in 2009, an RNN based architecture designed to recognize patterns won several contests by setting benchmarks for accuracy in handwritten data recognition.

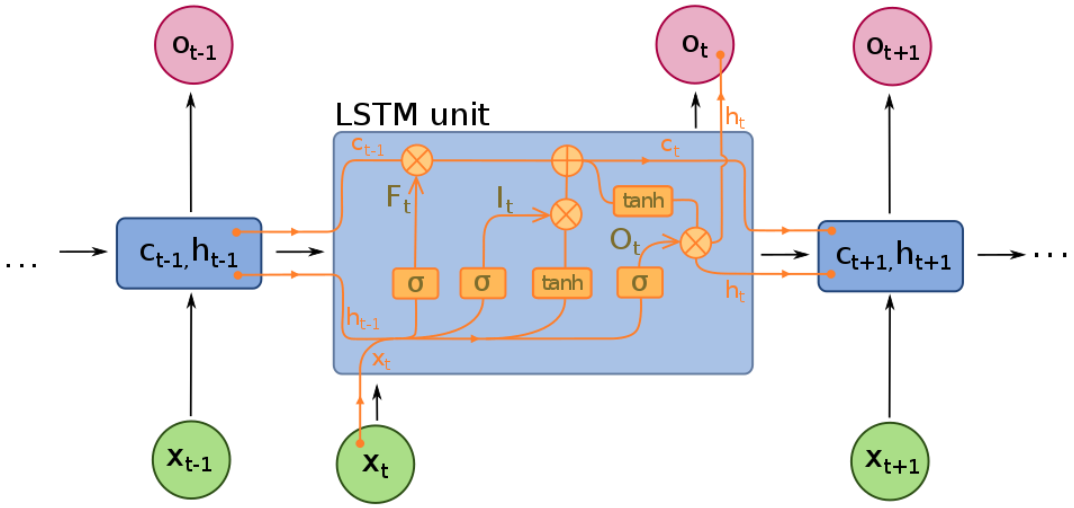


Figure 12 Long short-term memory

LSTM's are a deep learning architecture designed to avoid the vanishing gradient problem. The vanishing gradient problem is an issue that deep learning models face during training that arises due to loss of accuracy during backpropagation. Backpropagation allows each

of the networks weights to update slightly depending on training progress, but sometimes the gradient will be vanishingly small, prevent weights from changing values which may lead to the neural network to stop training entirely. Backpropagation initially discouraged researchers when they tried to make neural networks from scratch, and the vanishing gradient problem gave poor results, till the problem was correctly identified and possible solutions were invented, one of which are LSTM's.

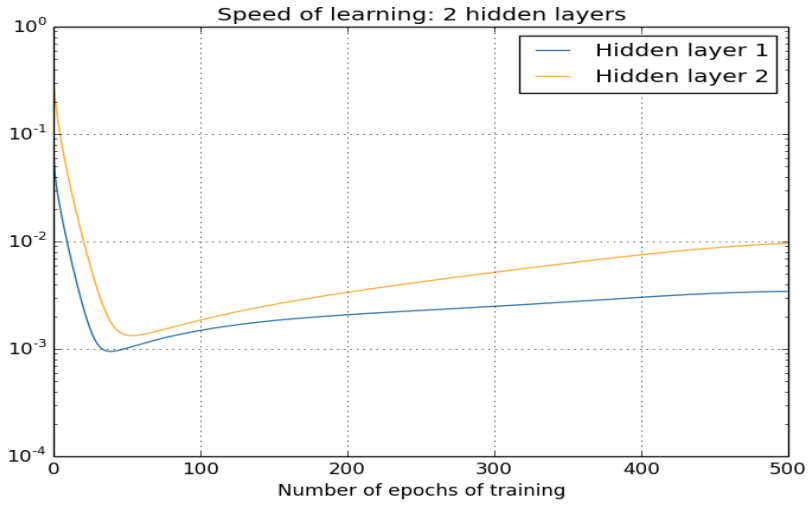


Figure 13 Vanishing Gradient over Layers

LSTM's are augmented by recurring gates which are called "forget" gates [22]. It prevents backpropagated errors from the vanishing gradient problem, which gives LSTM's freedom to learn tasks that require memories of events which happened millions of time steps earlier. LSTM's are also very specific and can be modified for specific tasks depending on the need of it.

Training an RNN

The steps required to training an RNN are very similar to training a CNN like we covered in the previous chapter. The concept of backpropagation is also the same, except there are slight modifications to the entire process itself. Unlike a CNN, the parameters between layers does not change in an RNN, so the parameters are shared by all time steps in the network, the gradient at each output depends not only on calculations on current time step but also previous time steps [20].

Training an RNN at a certain time step requires the calculation of gradient *n* steps behind it. This process is called Backpropagation Through Time (BPTT), which is underlying principle behind RNN architectures. Traditionally vanilla RNN's suffer from vanishing/exploding gradient problem, but specialized RNN's (LSTM) exist which do not suffer from that problem.

Extensions to RNN

RNN's are modifiable to be suited to perform specific tasks with high accuracy. One of the variations of which we discussed above known as the LSTM. Let's look at a few more popular variations of a vanilla RNN.

Bidirectional RNN's: Bidirectional RNN's are rooted on the idea that the output at a certain time step may be dependent not only on previous time steps but also on future time steps. Let's look at the example of predicting a missing member of a sequence. The RNN requires knowledge of previous time steps and future time steps to make a prediction. Bidirectional RNN's are two RNN's working on top of each other.

Deep RNN's: Deep RNN's are like Bidirectional RNN's, but with more layers per time step. These types of architectures take a higher learning capacity, more data and more time to train but can be used to model architectures for longer sequences. Like Deep RNN's, Wide RNN's are neural networks with more nodes rather than layers per time step.

Chapter V

Sign Language Dataset

The dataset used was a custom American Sign Language Dataset based off an existing dataset curated by Needell *et al.* [23]. The authors of the Neidel dataset collected data from recording ASL native signers under supervision of Neidel. Video stimuli was presented to the signers, and they were asked to produce the sign as they naturally would. The video stimuli also included variations of a sign which existed in the dictionary. The signers did not always produce the same sign however, there was variation in the signs produced. The variations were classified and labelled accordingly.

The signers were recorded using four synchronized cameras, providing a side view, a close and higher resolution view. Video processing was applied to ensure the frame rate over the videos stay the same and the resolution was high enough that the signs performed were legible to be recognized.

A custom dataset was used because of several drawbacks present in the original dataset. The presence of multiple interpretations of a single sign gesture led to lower accuracy of the model because of wrong features being extracted during the training process. The presence of variation in clothing and facial expressions also led to lower accuracy due to a lot of irrelevant features being trained to the model decreasing accuracy greatly.

We decided to use the most common 150 words in American Sign Language as a starting point. Each word was individually recorded 4 times (3 for test, 1 for training) whilst keeping the same clothes and removing facial features. The variation between multiple gestures accounted for variation in practical scenarios.

Figure 14 An Example from the Custom Dataset

The videos were recorded on an iPhone 6 camera at 60 frames per second. The videos were all recorded at a constant 720p resolution. Each video was then preprocessed before feeding to the model. Preprocessing the model involved breaking down each video to a size of 300 frames, which was used as the standard length of gestures. The dataset was then augmented to increase the size of training and test for better performance in training. The completed dataset will be available online without augmentation.

Id	Gesture	Id	Gesture	Id	Gesture	Id	Gesture
1	Accept	46	Breakfast	91	Father	136	Mad
2	Again	47	Brother	92	Fine	137	Marry
3	Beach	48	Buy	93	Friend	138	Meat
4	Bear	49	Busy	94	Fish	139	Milk
5	Become	50	Candy	95	Food	140	Mother
6	Blood	51	Cheese	96	Friday	141	More
7	Care	52	Clean	97	Change	142	Math
8	Certify	53	Coffee	98	Good	143	Me
9	Disconnect	54	Cold	99	Grandfather	144	Mean
10	Hit	55	Cookie	100	Grandmother	145	Mushroom
11	More	56	Cousin	101	Grapes	146	Monkey
12	Like	57	Cute	102	Green	147	Niece
13	Help	58	Cat	103	Garage	148	Nephew
14	Animal	59	Cow	104	Girl	149	Nice
15	Bird	60	Child	105	Girlfriend	150	Meaning
16	Cat	61	Car	106	Granddaughter		
17	Cow	62	Carrot	107	Grandson		
18	Elephant	63	Cherry	108	Green Beans		
19	Horse	64	Chicken	109	Grey		
20	Lion	65	Children	110	Gross		
21	Black	66	Dispatch	111	Giraffe		
22	Finish	67	Coke	112	Go		
23	Apple	68	College	113	Happy		
24	Aunt	69	Color	114	Hamburger		
25	Angry	70	Cook	115	Hot		
26	Apartment	71	Drink	116	Hurt		
27	Baby	72	Daughter	117	Husband		
28	Bad	73	Delicious	118	Help		
29	Banana	74	Divorce	119	Horse		
30	Beautiful	75	Date	120	Home		
31	Bicycle	76	Deaf	121	Hot dog		
32	Blue	77	Depressed	122	House		
33	Boy	78	Different	123	Hungry		
34	Bread	79	Dining Room	124	Ice Cream		
35	Brown	80	Dinner	125	Jam		
36	Butter	81	Door	126	Ketchup		
37	Bird	82	Dog	127	Kitchen		
38	Better	83	Eat	128	Know		
39	Bachelor	84	Exciting	129	Lemon		
40	Bacon	85	Big	130	Lunch		
41	Bald	86	Eggs	131	Like		
42	Bathroom	87	English	132	Lion		

43	Bedroom	88	Enter	133	Lettuce		
44	Bitter	89	Family	134	Living Room		
45	Boyfriend	90	Fruit	135	Love		

Table 1 List of Gestures

Chapter VI
Methodology and Experimental Results

Two main concepts the RNN and Inception (CNN) have been introduced and discussed in detail in previous chapter. CNN is focused on temporal feature extraction and the RNN is more focused on sequence recognition. In this chapter, we will connect the two fields and apply the technique of CNN to recognize individual gestures from images, and the recognize the sequences of images using an RNN.

Objectives

There are several goals that we will accomplish in this chapter.

- Examine the process of extracting images from gesture videos.
- Obtain the results of the CNN with predicted labels and the output of the pool layer.
- Build an RNN framework.
- Pass the labels to the RNN framework and train the RNN.
- Predict results.

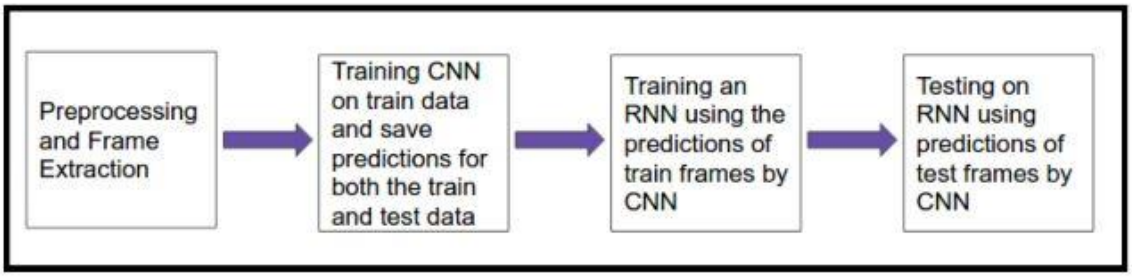


Figure 15 Experimental Process

Retraining Inception Model

In practice its hard to train a CNN from scratch, because it is rare to find a dataset to train it to the appropriate parameters. A solution to which is using a pretrained CNN as a starting point and use transfer learning. Several transfer learning scenarios exist such as [24]:

1. CNN as fixed feature extractor: Using a CNN which is pretrained and removing the last fully-connected layer, then use the rest of the CNN as a fixed feature extractor for the new dataset. Once we retrain the CNN we then use a classifier for the new dataset.
2. Fine-Tuning the CNN: The second strategy is to not only retrain the CNN but also modify the weights for the new dataset. It is possible to retrain every layer and modify or keep existing layers. This is used when earlier training of the model has generic features and we need to fine tune for specific features in the new dataset.
3. Pretrained model: Since newer models of CNN take several weeks to train across GPU's, it is common to use checkpoints for finetuning [19].

Deciding which transfer learning to use depends on several factors such as size of the new dataset and its similarity to the original dataset. There are several rules which help you decide [19]:

1. If the new dataset is small, you should not fine tune the CNN to avoid overfitting. It is a better idea to use higher level features instead of finer details.
2. If the new dataset is large and like the original training dataset then the CNN will have accuracy with good accuracy.
3. If the new dataset is small but also different from the original, using a linear classifier is the best idea, since CNN's pick up dataset specific terms.
4. If the new dataset is large and different we might have to retrain the CNN from scratch.

After reading every video in the dataset along with its corresponding gesture, we extract frames (pictures) from the videos of length 300. We ensure that the 300 frames capture all the important details from the corresponding gesture. After extracting frames and labelling images we run a retraining script to retrain Inception model discussed in the previous chapter. The top layer (bottleneck) is trained to recognize specific classes of images [25]. A SoftMax layer is used on the end to train for final class predictions.

Predicting Labels With CNN

Once the model is retrained, we predict labels for individual gestures by feeding it to the trained CNN. The process and the model are described below.

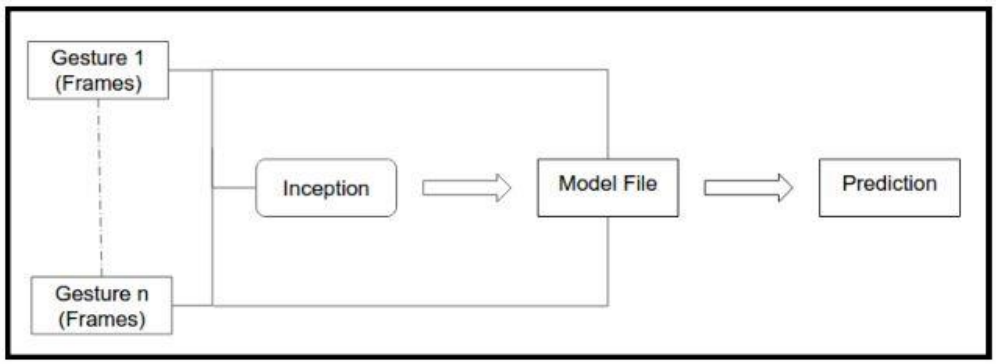


Figure 16 Model for predicting labels

If we take a gesture like *again*, the gesture gets broken into frames and each frame has an associated prediction and label attached to it.

The second approach to the problem was recording the output of the pool layer of the CNN instead of using the predicted label from the CNN. The preprocessing for both the approaches was the same, but the output varied and was stored independently.

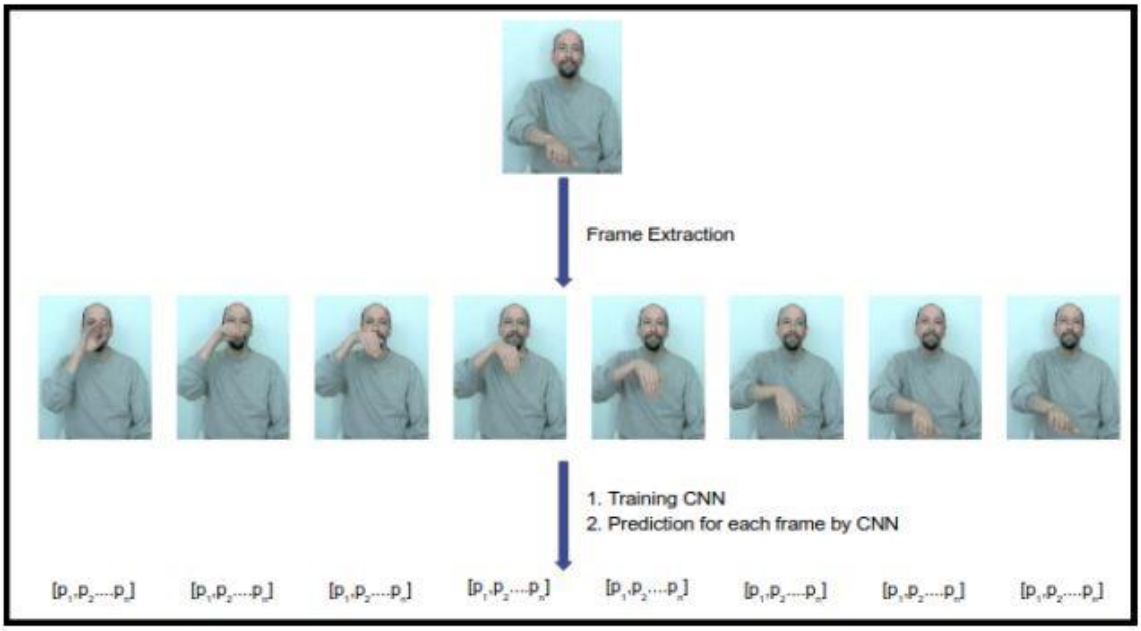


Figure 17 Frame Prediction for Gesture

Training and Setting up an RNN

We create an RNN model based on LSTM's which were discussed in the previous chapter. The first layer is used to feed input to the succeeding layers. The model we use is a wide network consisting of 256 LSTM units. The wide layer is followed by a fully connected layer with SoftMax activation. The size of the input layer is determined by the size of the input being fed, which in our case is 300. The fully connected layer is every neuron connected to every neuron of the previous layer and it consists of neurons equal to the number of classes. Finally, the model is finished by a regression layer to the input. We used ADAM [26], a stochastic optimizer as a gradient descent optimizer to minimize the provided loss function categorical cross entropy.

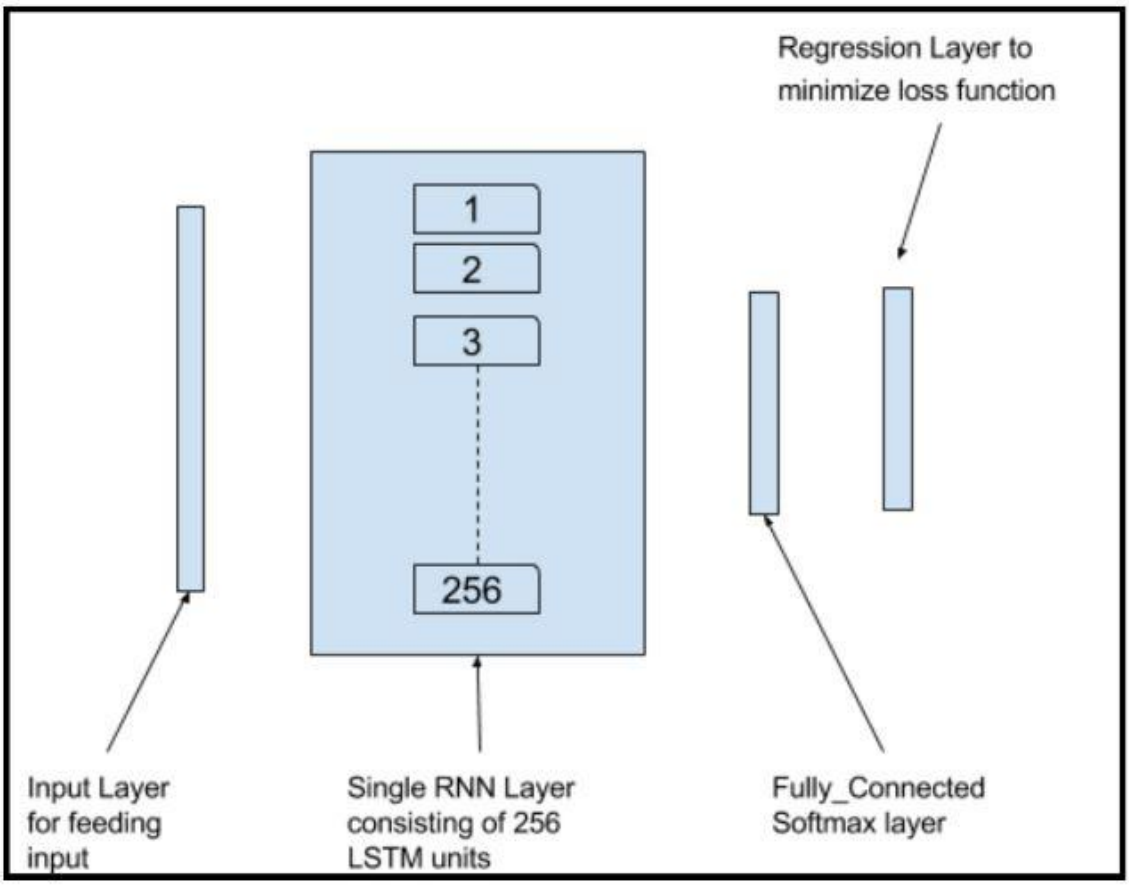


Figure 18 RNN Model

Other options we tested were using a wider RNN network with 512 LSTM units and a deeper RNN network with three layers of 64 LSTM units. After testing we concluded that the wide model with 256 units gave the best performance.

After defining the RNN model, we pass the video frames of both the approaches along with label information to the completed model for final gesture prediction.

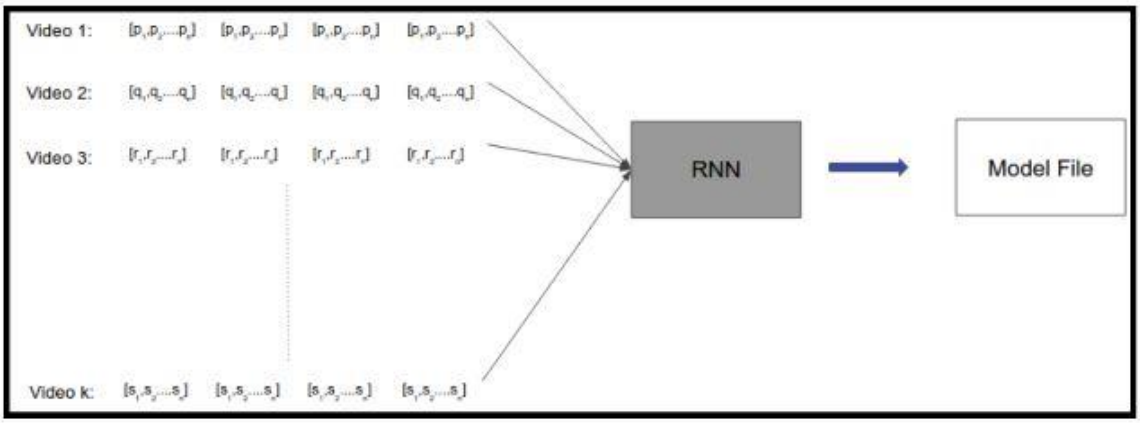


Figure 19 RNN Training

Results

The dataset used was the custom dataset recorded for this thesis, which has been described in the section above. It consisted of a total of 150 most common ASL signs. It was augmented to increase the number of images per class before training and testing. The augmentations consisted of resizing, expanding and shrinking the images. It also accounted for variation. The dataset consisted of videos, which were broken down to frames by using OpenCV in Python. The dataset was split into 80% for training and 20% for testing randomly.

The hyperparameters of the model like batch size of the model was set to 32 and with 10 epochs and a dropout of 0.3. ADAM was chosen to regularize for stochastic gradient descent. The hyperparameters were chosen after several runs and comparing the model accuracy. The model was also chosen after multiple experiments consisting of variation of

dropout, number of LSTM layers and number of LSTM nodes. During training of the model, a 10% split was used as validation set and was split at random from the training set.

The metric used to compare model performance was accuracy of class prediction by the model. We used two different approaches with the same model, comparing accuracies of two different outputs from the CNN to the RNN: a. Using the output of the final SoftMax layer i.e.. Final predictions and b. Using the output of the pool layer which is a 2048 sized vector.

For the sake of validation, we also compared the performance of the model to two existing models on SLR. Vivek [8] developed a model for American Sign Language recognition consisting of a custom CNN model consisting of 6 convolutional layers with a dropout of 0.25. and a final dropout layer of dropout 0.5. We extended that CNN model with the RNN we developed and trained it on our dataset. The original model was trained on a custom dataset of the authors based on the ASL dataset consisting of only static hand gestures. We also compared the model to a model developed by Lu [7] on SLR. The model consisted of a pretrained CNN named ResNet which was trained by transfer learning for the VIVA Gesture dataset followed by an RNN developed by the authors. The model was trained for 20 epochs with a learning rate of 1-e4 and ADAM for stochastic gradient descent. The batch size was set to 48 and 8-fold cross validation was used by the authors. The authors also performed augmentation on their dataset like ours. We trained the model on our dataset and compared the accuracy of the models with the same hyperparameters but without the cross validation.

The accuracy of the models stabilized with higher size of the dataset, although using the pool layer approach yielded poorer results than using the prediction layer. We evaluate the

CNN and RNN independently using the same training and testing dataset this ensures that the test data has not been seen by the CNN and the RNN. The models did not use cross validation for training.

# of Signs	Accuracy with SoftMax Layer	Accuracy with Pool Layer	Accuracy of Vivek et al. [8]	Accuracy of Lu et al. [7]
10 (20 samples)	75%	55%	82%	82%
50 (100 samples)	87%	58%	84%	81%
100 (200 samples)	91%	58%	84%	83%
150 (300 samples)	90%	55%	81%	83%

Table 2 Final Results on Testing Dataset

From the results we can see that the performance of the model improves with increase in the dataset size. The pool layer approached showed lower accuracy could possibly be because of conflicting features or too many features in the training set. The model performed these results on a dataset which consisted of well-lit subjects with minimal motion. If the subject were to have too much haphazard motion, the accuracy of the model would suffer.

The accuracy of misclassified signs did not correct with an increase in sample size, in fact originally correctly classified signs were later misclassified when increasing the number of signs which leads to a conclusion that there could possibly be too little difference between those signs for the model to differentiate and we need more features or more distinction to have better accuracy.

Chapter VII

Conclusions and Future Work

In this paper we introduced a way to recognize American Sign Language using machine learning. It is an approach to solve the problems faced by people with hearing and speech impairments. Its composed of 2 major components, analyzing the gestures from images and classifying images. Since we are dealing with a smaller dataset, using a larger dataset may provide better results.

We investigated two approaches to classification: using the pool layer and using the SoftMax layer for final predictions. The SoftMax layer provided better results because of distinct features. The sheer number of features in a 2048 vector confused the network leading to poorer results.

Problems Faced by The Model

One of the problems the model faced was the presence of facial features and skin tones. While testing with different skin tones, the model dropped accuracy if it hadn't been trained on a certain skin tone was made to predict on it.

The model also suffered from loss of accuracy with the inclusion of faces. Faces of signers vary, which leads to model to train incorrect features from the videos. The videos had to be trimmed to include only gestures which were only extended to the neck of the signer.

The model also performed poorly when there was variation in clothing. Maybe using a ROI to isolate hand gestures from the images would help accuracy, but for the context of this paper, a consistent full-sleeved shirt was used in all the gesture recordings.

Potential Improvements

One of the potential improvements would be to experiment with different RNN architectures for the output of the pool layer. Including GRU and Independent RNN's.

In terms of CNN improvements, using Capsule Networks [27] instead of Inception may yield better results than Inception along with working on integrating the CNN and RNN model into one ensemble. Generally using two different models to feed into each other, suffers from loss of data and increase of training time, whereas using one ensemble allows for careful monitoring of input data and precise corrections to the model.

Chapter VIII

Bibliography

[1] V. Athitsos, C. Neidle, S. Sclaroff and J. Nash, "The American Sign Language Lexicon Video Dataset," *2008 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops,* 2008.

[2] J. Huang, W. Zhou and Q. Zhang, "Video-based Sign Language Recognition without Temporal Segmentation," *arXiv,* 2018.

[3] T.-W. Chong and B.-G. Lee, "American Sign Language Recognition Using Leap Motion Controller with Machine Learning Approach," *Sensors,* vol. 18, 2018.

[4] C. Linqin, C. Shuangjie and X. Min, "Dynamic hand gesture recognition using RGB-D data for natural human-computer interaction," *Journal of Intelligent and Fuzzy Systems,* 2017.

[5] F. Ronchetti, Q. Facundo and A. E. Cesar, "Handshake recognition for argentinian sign language using probsom," *Journal of Computer Science & Technology,* 2016.

[6] C. Hardie and D. Fahim, "Sign Language Recognition Using Temporal Classification," *arXiv,* 2017.

[7] D. Lu, C. Qiu and Y. Xiao, "Temporal Convolutional Neural Network for Gesture Recognition," Beijing, China.

[8] V. Bheda and D. N. Radpour, "Using Deep Convolutional Networks for Gesture Recognition in American Sign Language," Department of Computer Science, State University of New York Buffalo, New York.

[9] Y. LeCun, "LeNet-5, Convolutional Neural Networks".

[10] M. Masakazu, K. Mori, Y. Mitari and Y. Kaneda, "Subject independent facial expression recognition with robust face detection using a convolutional neural network," *Science Direct,* 2003.

[11] Y. LeCun, C. Cortes and B. Christopher, "MNIST handwritten digit database".

[12] J. Ludwig, "Image Convolution," *Portland State University.*

[13] "cs231n.github.io," [Online].

[14] A. Krizhevsky, I. Sutskever and G. Hinton, "Imagenet classification with deep convolutional neural networks," *Advances in Neural Information Processing Systems.*

[15] D. Scherer, A. Muller and S. Behnke, "Evaluation of Pooling Operations in Convolutional Architectures for Object Recognition," *20th International Conference on Thessaloniki, Greece,* 2010.

[16] N. Srivastava, H. Geoffrey and K. Alex, "Dropout: A Simple Way to Prevent Neural Networks from overfitting," *Journal of Machine Learning Research,* 1929-1958.

[17] "Regularization of Neural Networks using DropConnect," *ICML,* 2013.

[18] H. Sak, A. Senior and F. Beaufays, "Long Short Term Memory recurrent neural network architectures for large scale acoustic modelling," 2014.

[19] M. Milos, "Comparitive analysis of Recurrent and Finite Impulse Response Neural Networks in Time Series Prediction," *Indian Journal of Computer and Engineering,* 2012.

[20] D. Britz, "Recurrent Neural Networks Tutorial, Part 1 – Introduction to RNNs," WILDML, 17 September 2015. [Online]. Available: http://www.wildml.com/2015/09/recurrent-neural-networks-tutorial-part-1-introduction-to-rnns/.

[21] J. Schmidhuber, "Habilitation thesis: System modeling and optimization," *Credit assignment across the equivalent of 1200 layers in an RNN,* p. 150.

[22] F. Gers, N. Nicol and J. Schmidhuber, "Learning Precise Timing with LSTM Recurrent Networks," *ResearchGate,* p. 143, 2017.

[23] C. Neidle and A. Thangali, "Challenges in Development of the American Sign Language Lexicon Video Dataset Corupus," *5th Workshop on the Representation and Processing of Sign Language,* 2012.

[24] "http://cs231n.github.io/transfer-learning/," [Online].

[25] "https://becominghuman.ai/transfer-learning-retraining-inception-v3-for-custom-image-classification-2820f653c557," [Online].

[26] Kingma, Diedrick and J. Ba, "ADAM: A method for stochastic optimization," *arXiv,* 2014.

[27] G. Hindon, "Dynamic Routing Between Capsule".