IJREAT International Journal of Research in Engineering& Advanced Technology, Volume 8, Issue 4, Aug - Sep, 2020
ISSN: 2320 – 8791 (Impact Factor: 2.317)
www.ijreat.org

AN APP BASED SOLUTION FOR SIGN LANGUAGE TO TEXT CONVERSION

[1]Adithi Krishnan, [2]Ruthvik B.R, [3]Spoorthy M, [4]Rhea Muthanna M, [5]Shashank N

1Department of Computer Science and Engineering,Vidyavardhaka College of Engineering, Mysore, Karnataka,India, adithikrishnan@yahoo.co.in

2Department of Computer Science and Engineering, Vidyavardhaka College of Engineering, Mysore, Karnataka, India, ruthvikbr24@gmail.com

3Department of Computer Science and Engineering, Vidyavardhaka College of Engineering, Mysore, Karnataka, India, spoorthymahesh183@gmail.com

4Department of Computer Science and Engineering, Vidyavardhaka College of Engineering, Mysore, Karnataka, India, rheamuthanna.muthanna@gmail.com

5Department of Computer Science and Engineering, Vidyavardhaka College of Engineering, Mysore, Karnataka, India, shashank.n@vvce.ac.in

***Abstract* --**Sign languages are a way in which languages use the visual – manual means to convey a message to the other person. It usually has a predefined sign with each symbol representing a letter. Series of signs are used in order to convey a sentence. The main aim of this paper is to develop a mobile application-based solution that takes sign language gestures as input to a trained deep learning model built using 2D Convolutional Neural Networks and converts it to text and voice outputs in real-time for improved and finer communication. The basic idea behind this is that sign language is not one that is commonly learnt by all and hence people who are familiar with sign language only are able to communicate with the mute. This solution aims to bridge that knowledge gap between people irrespective of their familiarity with sign language. After implementing our solution, it is found that the model predicts gestures with an accuracy rate of 78%. Once translated, the text was also converted to audio output using Text-to-speech library(tts). This app-based approach comes in handy as most people today use smartphones and hence the application can reach out to more users.

Keywords – Sign Language, Text, Voice, Convolutional Neural Network

1. INTRODUCTION

Language is a tool to solve the challenge of explicit communication. Under language, we have two types: Spoken Languages and Sign Languages. Spoken languages uses phonetics and sounds whereas Sign Languages mainly involve gestures. Translating these gestures to actual human understandable sentences in real-time is challenging due to various factors like lighting, skin tone, background clutter etc. This conversion is also a step by step process that involves image processing, object detection, computer vision, natural language processing etc. This section tries to summarize why this problem was researched, what each of the above technologies are and how they are used in developing the application followed by what are the exact steps taken in developing the application.

Sign Languages are the most important communication tools for deaf and hard-of-hearing people. It is also used by people who aren't deaf but cannot speak. It is more expressive than spoken languages and involves a lot of facial expressions, gestures, and hand movements. The same words can become a statement, or a question based on the facial expression. Hence, it is very challenging for a computer to interpret sign language gestures. Not often do we have a system that can recognize signs with a high rate of accuracy and hence sign language recognition and translation remains a field with ongoing research.

Computer Vision is a subfield of Artificial Intelligence that uses Image Processing algorithms to solve image related problems. Image Processing deals with the methods that apply operations on a given image to obtain an enhanced image or to obtain useful information from it using feature extraction. There are 2 types of image processing – analog and digital. We mainly focus on digital image processing which is used by modern digital devices/computers to process digital images and perform image analysis. The difference between computer vision and image processing can be summarized as follows: If we want to enhance an image or extract some information from it, say apply any transformations such as smoothening, sharpening etc., then it is plain image processing. If our goal is to mirror/emulate human vision such as detecting features, detecting objects, understanding the image, then it falls under Computer Vision. The application presented in this paper explains how to use Image Processing and Computer Vision techniques in taking the gesture in the form of an input image and processing it to remove any nonessential information.

Object detection is a key field that acts like a bridge between Computer Vision and Image Processing. It mainly focuses on detection of objects/instances of a class in a frame. It usually uses machine learning or deep learning techniques to furnish substantial results. Machine Learning approach usually involves defining features and then classifying the objects using some algorithm such as SVM or KNN classifier. Deep learning approaches perform object detection from start to end and make use of Convolutional Neural Networks. The methodology presented in this paper uses Convolutional Neural Networks to construct the deep learning model.

Deep learning is a sub branch of machine learning which uses layers of neural networks to obtain the important features from a raw image. The network may be Convolutional Neural Networks, Recurrent Neural Networks, Generative Adversarial Networks etc. In the methodology proposed, Convolutional neural network that uses multiple layers of perceptrons for analyzing the data plays an important role in building the model.

Natural Language Processing is another branch of Artificial Intelligence, a platoon of techniques which bestow upon computers the potential to read and infer from human languages. These techniques are mainly used in human – machine interactions. The application presented in this paper has leveraged the text to speech conversion technique under this field to convert text output generated to audio output as well.

The application presented in this paper has been designed to recognize gestures of the American Sign Language. The approach used to achieve this is as follows:

1. The dataset used is the ASL Alphabet dataset downloaded from Kaggle.
2. A 2D Convolutional Neural Network with an input layer, 5 hidden layers and one output layer is used for training.
3. Keras, which runs on TensorFlow backend is used to implement the network and train the network on the training dataset. A ".h5" file is obtained after training.
4. This h5 file is then converted to a 'tflite' (TensorFlow Lite) model which is used in the Android application.
5. The android application uses the camera present in the phone to take pictures of gestures as input.
6. This image is then passed to the tflite model which returns the 3 most likely translations of that gesture.
7. Once the desired translation is selected, it is displayed as text on the screen.
8. To convert the same text to audio, we have leveraged the TTS (text to speech) library available in Android which converts the text to audio and plays it out loud.

A detailed explanation of each of the above steps in the process is present under the Methodology section

2. BACKGROUND AND RELATED WORK

Based on our initial survey regarding various techniques used for sign language translation as presented in [1], we feel, most of the approaches make use of Convolutional Neural Networks in building the trained model. The methods which present real-time detection of sign language gestures usually take video as input, divides it into frames and converts each frame into its corresponding sign using the model. Some methods have also used SVMs and LPP algorithms for real time sign language to text conversion. There is also another approach which involves the use of flex sensors attached to gloves that can be worn on the hand. Based on the hand movements, the sensors pick up various gestures which are then converted to text using an analog to digital converter and microcontroller. Since Convolutional Neural Networks form the basis for many of the approaches presented, we decided to deep dive into it and research a bit more to understand and build a network that adapts to our needs.

2.1 NEURAL NETWORKS

Neural networks are circuits of neurons, or as more frequently expressed, a network of artificial neurons/nodes. These networks are constructed in such a way as to mimic a biological neural network and the connections of a biological neuron are modeled as weights. They are usually used to identify relationships between vast amounts of data, usually unstructured data. The neural is a network that has weights on it and we can adjust the weights in order to train the network. This happens with a lot of trails and fine-tuning parameters. The following image shows the many types of neural networks used.

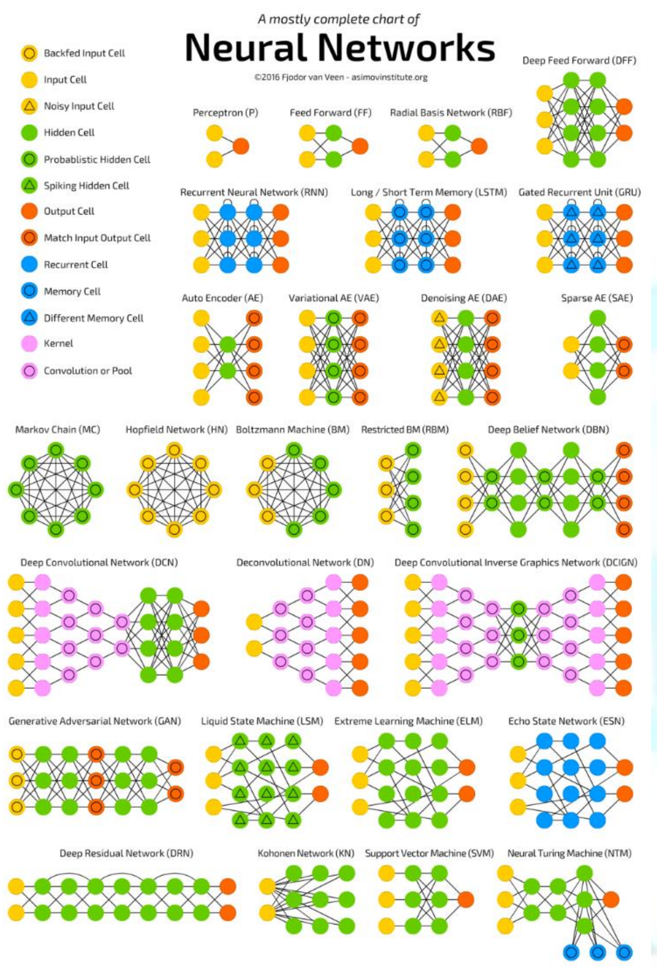
A mostly complete chart of
Neural Networks
©2016 Fjodor van Veen - asimovinstitute.org
Backfed Input Cell
Input Cell
Noisy Input Cell
Hidden Cell
Probablistic Hidden Cell
Spiking Hidden Cell
Output Cell
Match Input Output Cell
Recurrent Cell
Memory Cell
Different Memory Cell
Kernel
Convolution or Pool
Perceptron (P)
Feed Forward (FF)
Radial Basis Network (RBF)
Deep Feed Forward (DFF)
Recurrent Neural Network (RNN)
Long / Short Term Memory (LSTM)
Gated Recurrent Unit (GRU)
Auto Encoder (AE)
Variational AE (VAE)
Denoising AE (DAE)
Sparse AE (SAE)
Markov Chain (MC)
Hopfield Network (HN)
Boltzmann Machine (BM)
Restricted BM (RBM)
Deep Belief Network (DBN)
Deep Convolutional Network (DCN)
Deconvolutional Network (DN)
Deep Convolutional Inverse Graphics Network (DCIGN)
Generative Adversarial Network (GAN)
Liquid State Machine (LSM)
Extreme Learning Machine (ELM)
Echo State Network (ESN)
Deep Residual Network (DRN)
Kohonen Network (KN)
Support Vector Machine (SVM)
Neural Turing Machine (NTM)

Fig 2.1.1 The types of neural networks [2]

Some terms associated with neural networks that we need to understand the definitions of are as follows:

- **Kernel** – It is a convolution matrix or mask which is used for blurring, edge detection , sharpening and more by convoluting a kernel and an image.
- **Kernel Size** –It determines the size of the kernel. Common dimensions of the kernel size are (1x1), (3x3), (5x5) and so on. It specifies the height and width of the kernel and it must be of odd integer sizes only.
- **Max Pooling** – Replacing each patch in the input with a single output is known as pooling. Max pooling refers to moving a 2D window across the image, where the maximum value within each window is given as output.
- **Flatten** – It is used on tensors to remove all its dimensions except one. This helps in reshaping tensors so that it will have the shape equal to the number of elements in the tensor, not including the dimension of the batch.
- **Dropout** – During the training neurons are randomly chosen to be ignored. They are ignored during a particular backward/forward pass. Dropout is not applied to the output layer of the neural network.
- **Dense** – It is an operation that connects every input to every output by a weight. It is followed by an activation function that is not linear.
- **Activation** – These functionsdetermine the output of a neural network and are basically a mathematical equation. Each neuron in the network is associated with this and the output of this function determines if a neuron should be activated / fired or.
- **Sigmoid** - It is an activation function used in neural networks whose value lies between 0.0 and 1.0 indicating the probability of a classification.
- **ReLU** – Rectified Linear Unit isdefined mathematically as y = maximum(0, x). The function is linear for all positive inputs and 0 for negative. It is one of the Most used activation functions in CNNs and is cheap to compute and training time is also less.
- **Softmax** - This functionis used for multiple classes. It gives the probabilities for each class in the range of 0 and 1. The sum of all the classes are calculated and used to divide the previously obtained probabilities. This gives the probability of an input belonging to each class. It is useful for output neurons. It is used only for the output layer, for classification with more than 2 classes.
- **Kernel initializer** - It is the initializer for the kernel weights matrix.
- **KernelRegularizer** - It is a technique used to reduce overfitting by fitting a function appropriately on the given training set
- **Batch Normalization** - It is a technique used to training DNNs that standardize the inputs to a layer for each mini-batch
- **Padding** – It is a parameter that determines whether the kernel is zero-padded or not. If it is not zero padded, it means that the spatial dimension can reduce via the application of convolutions.
- **Optimizer** – Optimizers are methods used to change attributes of a neural network to minimize the losses.
- **Adam Optimizer** - It is used in training deep learning models. This optimizer is an optimization algorithm for stochastic gradient descent . Adam combines the pros of the AdaGrad and RMSProp algorithms making it easy to handle sparse gradients on noisy problems.

2.2 CONVOLUTIONAL NEURAL NETWORKS

CNNs fall under the category of deep neural networks. They are most commonly used for image analysis, image recognition, object detection etc. They are specifically designed to process pixel data and it is also observed that the CNNs have their neurons arranged similar to those in the brain's frontal lobe, the area which is responsible for handling visual input in animals. Traditional neural networks are usually given pieces images with reduced resolution as input but given their architecture, CNNs can be given high resolution images thus avoiding the above problem.

The following image depicts a typical Convolutional Neural Network used for image classification.

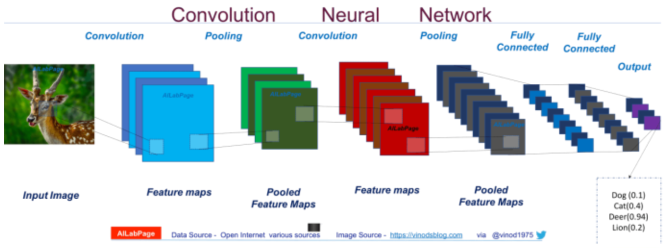

Fig 2.2.1 Typical Convolutional Neural Network [3]

2.2.1 ARCHITECTURE OF CONVOLUTIONAL NEURAL NETWORK [4]

This section explains how a Convolutional Neural Network trains and predicts and is based on the mathematical operations/logic behind it.

A Convolutional Neural Network usually takes a tensor of order 3 (an image with R rows, C columns and D color channels where D is usually 3 (RGB) in case of color images. Higher order tensors can also be handled by CNNs. Since we are dealing with color images in this problem, we will continue with an order 3 tensor as input. Each tensor/image then undergoes a series of processing steps sequentially. Each processing step here is called a layer.

$$\boldsymbol{x}^1 \longrightarrow \boxed{\boldsymbol{w}^1} \longrightarrow \boldsymbol{x}^2 \longrightarrow \cdots \longrightarrow \boldsymbol{x}^{L-1} \longrightarrow \boxed{\boldsymbol{w}^{L-1}} \longrightarrow \boldsymbol{x}^L \longrightarrow \boxed{\boldsymbol{w}^L} \longrightarrow z$$

Fig 2.2.1.1: The equation of a Convolutional Neural Network

The above image illustrates the equation of a Convolutional Neural Network, how the network runs layer by layer. Each box depicts each layer and the input image undergoes processing in every layer. Parameters involved in the processing at each layer are collectively depicted as w^1 to w^{L-1}. One additional layer is added for backpropagation. The last layer is a loss layer and usually it is a cross entropy loss for classification problems.

2.3 INTRODUCTION TO ANDROID APPLICATION PROGRAMMING [5]

Android is one of the prominent Operating Systems for mobiles devices. Developing applications for Android devices lets us target a large audience. Android applications are developed using Android studio. We can make use of Java or Kotlin to build and deploy applications. We can also use the inbuilt tools to test and integrate with cloud. We have built an Android application which targets the majority of the devices from Android version “Lollipop” till Android version “pie”.

Fragments are a small component that act like building blocks of a layout and can be used to alter layouts according to device specification. Same fragments can be used to design different layout viewing for tablet devices and mobile devices. Fragments run in the context of an activity and they contain certain reusable functionalities. We have made use of 2 fragments in our application.

2.4 INTRODUCTION TO KERAS [6]

Keras is an open source neural network API. It is built using Python and can be used on top of TensorFlow, CNTK andTheano. It allows fast and easy prototyping, supports both CNNs and RNNs as well as a combination of both. It is designed to run coherently on both CPUs and GPUs. User friendliness, Modularity, Easy extensibility and working with Python are the main advantages of Keras over other libraries.

We have used Keras version 2.3.1 running on a TensorFlow backend while building the model.

2.5 INTRODUCTION TO TENSORFLOW LITE [7]

TensorFlow Lite is an open source framework for deep learning and helps us to run deep learning models on mobile, embedded and IoT devices. It helps us leverage the power of on-device learning with low latency and small binary size. It consists of two components – TensorFlow Lite converter and interpreter. The interpreter runs the optimized Machine learning models on different devices and the converter is used to convert TensorFlow models into a format like tflite thatwill be used by the interpreter.

Using TensorFlow Lite has advantages that include reduced latency, privacy, connectivity and reduced power consumption. Since models are converted and stored on the device, there is no round trip needed to get responses from model stored on a server. This reduces latency. Data that is fed to the model need not leave the device and hence this takes care of data privacy. Since no network connectivity is needed, this reduces the power consumed by the application.

The TensorFlow Lite binary requires only around 300KB when using operatorsfor supporting pre-trained image classification models like MobileNetand InceptionV3. If passing all parameters is required, then the size may go up to a maximum of 1MB. For our application, we have converted the generated h5 file to a TensorFlow model and then to a TensorFlow Lite model.

3. METHODOLOGY

The solution method adopted by us is to first train the network on the ASL Alphabet Dataset downloaded from Kaggle, convert the obtained model to a TensorFlow Lite model to enable on device inference. Next is to build an Android application that uses the in-device camera to obtain sign language gestures and send it to the model for translation. Once the translation is obtained, we display the text on screen. This text string is also converted to audio output using the Text to Speech library present in Android. The audio output will then be fed to smart assistants such as Alexa, google assistant which will make it easier to solve user requirements.

The following section gives a detailed explanation about how the application and neural network are built.

3.1 THE DATA SET USED

We have downloaded the “ASL Alphabet” data set from Kaggle. This dataset consists of 2 folders: ASL Alphabet Train and ASL Alphabet Test. The Train folder consists of 29 subfolders, one for each letter of the

Alphabet and one each for “del”, “space” and “nothing”. Each subfolder consists of 3000 images for each letter, taken at different angles, with varied lighting and at different proximities as well. The Test folder consists of 29 subfolders, one for each letter of the Alphabet and one each for “space”, “del” and “nothing”. Each of these subfolders contains 1 image each. All the images are of size 200x200.

3.2 THE CONVOLUTIONAL NEURAL NETWORK

We have 87000 images in the training directory which is 3000 per class and 8700 images in the validation directory which is 300 images per class. The training is run for 50 epochs. The training images are rescaled by a factor of 1/255, rotation range of 30, shear range and zoom range of 0.3, width and height shift range of 0.4 with random horizontal flipping. Images in the validation category are also rescaled by a factor of 1/255.

We have leveraged the Keras library to build the CNN to train the model using the Keras*Conv2D()* function and the *Sequential()* class. The *Conv2D()* function adds a 2D convolution layer that applies a spatial convolution on an input image and outputs a tensor.

We have built a CNN with 7 blocks. The batch size of the network is set to 100 and the kernel initializer is set to “he_normal” which obtains samples from a truncated normal distribution centered on 0.

The first block is made up of a 2D layer with 32 filters, kernel size of (7,7), and uses ReLU activation function and batch normalization. This is followed by another 2d convolution layer with 32 filters, 7x7 kernel size and “ReLU” activation and batch normalization. A max pooling layer with pool size (2,2) is added along with a dropout of 0.2.

The second, third and fourth blocks consist of the same layers with changes only in the number of filters and kernel size. Block 2 has 64 filters with kernel size (5,5), Block 3 has 128 filters with kernel size (3,3) and Block 4 has 256 filters with kernel size (3,3).

Block 5 first flattens the output of the 4th block and then has a dense fully connected layer with 64 filters. ReLU activation function with batch normalization and dropout of 0.5 is added. Block 6 is like block 5 without the flattening performed.

Block 7 is the output layer which has a fully connected/dense layer with 29 neurons and a Softmax activation function.

Early Stopping and ReduceLROnPlateau are also leveraged to reduce overfitting. EarlyStopping is a callback that allows us to monitor the performance measure and the mode. Once triggered, it will stop the training process. We have set the monitor to minimize validation loss. ReduceLROnPlateau is another callback which reduces the learning rate when a particular metric does not improve further. Monitor is set to validation loss with a reduce factor of 0.2 and a threshold learning rate of 0.0001. We have also used the Adam optimizer with a learning rate of 0.01 to minimize the “categorical cross entropy” loss.

The checkpoints for the model with monitor as validation loss are stored as a h5 file with only the best weight values stored at the end of training in the model file.

This model file is then converted to a tflite model using TensorFlow library. This file will be stored in an “assets” folder in the project file. Storing here would mean that the android application will use the tflite model file in its raw form without compressing it. This file will be built into the APK.

3.3 THE ANDROID APPLICATION

The application makes use of the mobile camera and displays everything that can be viewed through the camera in the camera preview built in the app. This preview in turn sends frames one by one to the classifier. The job of the classifier is to detect the part of the image that contains the hand and then send it to model as input. The input size of the image for the model is 200 x 200.

The classifier then receives the top three outputs from the model, and these are displayed under the preview. The next step is to form a sentence, which can be done by using the add button provided below the results. This button adds the result with the highest probability. The user uses sign language gestures to form sentences before the camera and forms the sentences. Once the sentence is formed, the user will click on the speak button which will send the string output to the ‘tts’ library and ‘tts’ methods give us the audio output. This output will be eventually fed to assistants like Alexa, Google Assistant. The application also works when the device is rotated.

The application is designed using two fragments. The first fragment is a camera preview that displays the live feed through the camera of the device. We make use of “camera 2 API”. The second fragment is a layout that displays the results and contains a button to add words to form a sentence and then another button to pass the formed sentence to the “Text to Speech” methods, thereby obtaining the audio output. Both fragments are then used one below the other in one activity.

4. RESULTS AND DISCUSSION

With the above training model in place, after training for 15 epochs, we were able to achieve a loss of 0.4083 or 40.83%, a training accuracy of 0.8748 or 87.48%, a validation loss of 0.1045 or 10.45% and a validation accuracy of 0.9681 or 96.81% when using “ReLU” activation function for all blocks and “Softmax” function for the output layer. We also tested the same model with the “sigmoid” activation function in combination with “Softmax” at the output layer, results for which are discussed below.

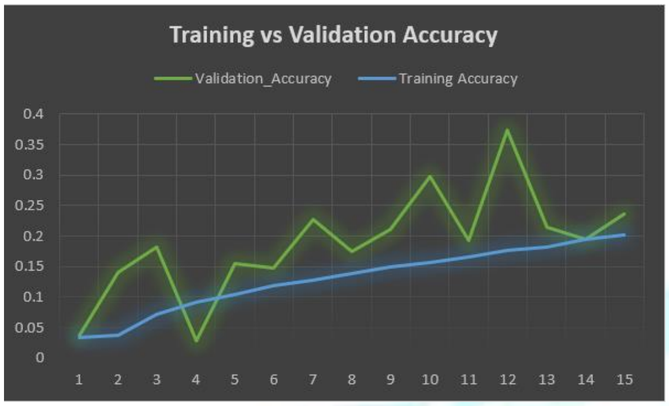

Fig 5.1 Training vs Validation Accuracy (sigmoid)

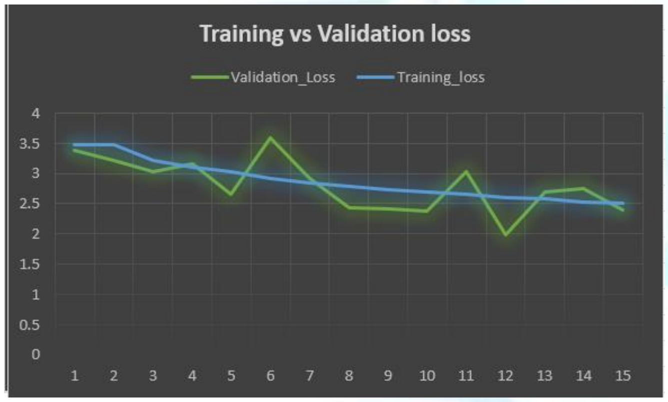

Fig 5.2 Training vs Validation loss (sigmoid)

Due to the callbacks present in the code, i.e., early stopping and reduceLROnPlateau, the training ended at 15 epochs and did not continue further.

The following graphs represent the variation in the above 4 parameters between epochs when ReLU activation function is used.

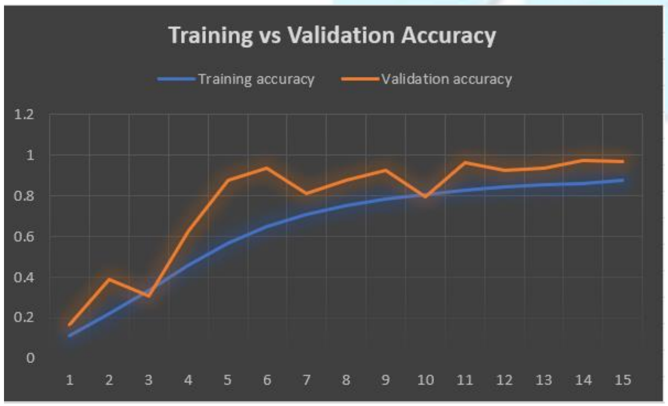

Fig 5.3 Training vs Validation Accuracy (ReLU)

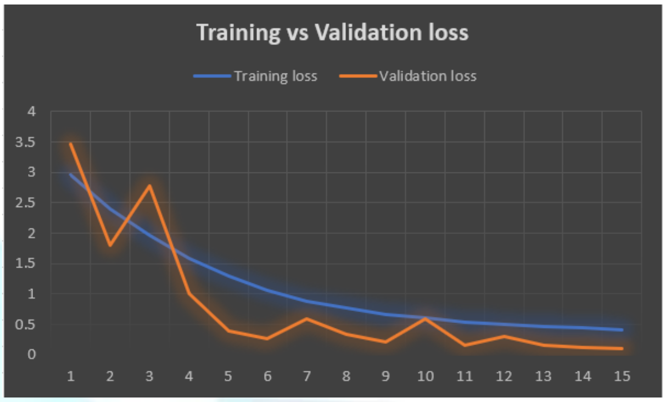

Fig 5.4 Training loss vs Validation loss (ReLU)

The graphs presented above depicts that there are a lot of variations in the training vs validation metrics when sigmoid is used whereas not much variation is present when ReLU function is used. The model also performs better with "ReLU" rather than "sigmoid".

We have tested our application with different input gestures and with varying background and lighting. After analyzing the outputs, we feel it is safe to say that our application can effectively translate ASL gestures to text and audio with an accuracy rate of 80%.

There are certain drawbacks in our application. The application might not work as expected in crowded areas or regions with less/poor lighting conditions. The current application works with a local model and hence is trained to predict only American Sign Language gestures. We can host the model in Firebase and use multiple models according to the different regions which use different sign language.

One more feature which we plan to incorporate in our application is the exact reverse of what we are doing, i.e., take text or audio as input and output it on the screen in the form of sign language gestures.

These are some points which we can take for further improvement of our application and for future research.

5. CONCLUSION

As related to the problem statement mentioned in the "Abstract" section, we have tried to develop an Android application that takes in American Sign language gestures as input and converts them to text and audio output in real-time. Using a CNN to train the model on the dataset has given us an accuracy of 86% with which we are able to classify the gestures real-time.

TensorFlow Lite works exceptionally well for devices with less processing powers. If we want to host the model using Firebase, we need network connectivity to download the model. This is eliminated in our application as we are using an inbuilt model. APK size is 24MB. Our application also provides links to tutorials on American Sign Language so that everyone can learn sign language. We feel that this application is helpful for developing a more connected society where language and speech is not a barrier for people to communicate effectively. This application would help people to understand what the mute are trying to say or express using their gestures and understand them better.

Future areas of research would include methods to improve the accuracy by trying image augmentation and transfer learning, converting text and audio back to sign language gestures and how to effectively make use of neural networks for this task.

6. ACKNOWLEDGEMENT

The authors express their appreciation for the support provided by our mentors and faculty members who have guided us during the research and helped us achieve desired results.

7. REFERENCES

[1] Adithi Krishnan, Ruthvik B R, Spoorthy M, Rhea Muthanna M, Shashank N, "*Sign Language to Text Conversion – A Survey*", International Journal of Scientific and Engineering Research, Volume 8, Issue 1, November 2019. ISSN 2229-5518.

[2] https://www.digitalvidya.com/blog/types-of-neural-networks/

[3] https://vinodsblog.com/2018/10/15/everything-you-need-to-know-about-convolutional-neural-networks/

[4] Jianxin Wu, "*Introduction to Convolutional Neural Networks*", LAMDA group, National Key Lab for Novel Software Technology, Nanjing University, China, May 1, 2017.

[5] http://developer.android.com/

[6] https://keras.io/

[7] https://www.tensorflow.org/lite/guide