American Sign Language Detection

Hiten Nirmal
University of Georgia
Department of Computer Science
hn97292@uga.edu

Ankit Vaghela
University of Georgia
Department of Computer Science
ankitkumar.vaghela@uga.edu

Gaurav Agarwal
University of Georgia
Department of Computer Science
gaurav.agarwal1@uga.edu

Abstract

Deep Learning has aided Computer Vision problems immensely. American Sign Language has been very important research problem in Computer Vision community. A great obstacle to American Sign Language Detection problem is the lack of large annotated data-set which works for real time web-cam images for various backgrounds. This technology should predict American Sign Language gestures directly from image pixels to help disabled people communicate via web-cam based chat platforms. This project is our trial to build such a technology. We implemented a web based client-server framework where we detect hand position using Tensor flow Object Detection API and send the cropped hand image to our classifier which predicts correct gesture. Our project uses ASL Alphabet data-set from Kaggle which contains 3000 images for each alphabet including "Space", "Delete" and "Nothing" labels. [1]

1 Introduction

American Sign Language (ASL) is a complete, complex language that employs signs made by hand gestures. It is the primary language of many North Americans who are deaf and is one of several communication options used by people who are deaf or hard-of-hearing. Sign languages are based on the idea that vision is the most useful tool a deaf person has to communicate and receive information. [2] The alternative of written communication is cumbersome, impersonal and even impractical when an emergency occurs. In order to diminish this obstacle and to enable dynamic communication, we present an ASL recognition system that uses Convolution Neural Networks (CNN) in real time to translate a video of a user's ASL signs into text. [3]

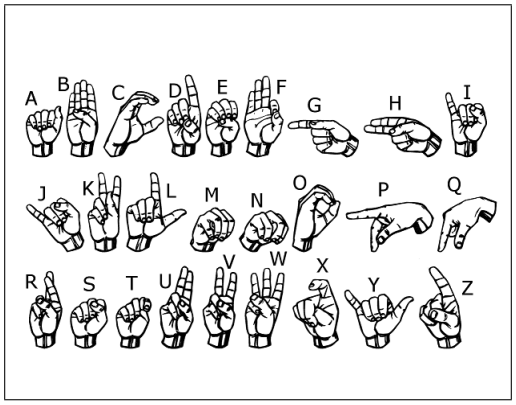


Figure 1: American Sign Language

Submitted to CSCI 8955 Advance Data Analytics: Statistical Learning and Optimization, University of Georgia on 11th December 2018. Do not distribute.

Basic challenges in Sign Language Detection are

- Environmental concerns (e.g. lighting sensitivity, background, and camera position)
- Occlusion (e.g. some or all fingers, or an entire hand can be out of the field of view)
- Sign boundary detection (when a sign ends and the next begins)
- Co-articulation (when a sign is affected by the preceding or succeeding sign)

1.1 Existing Approaches

We researched how this problem is solved by some existing approaches and we found two basic approaches

- **Static Bounding Box approach:** In this approach, a static bounding box is drawn over the web-cam image. User would place his/her hand in the static box and then this static box is cropped and passed on to the classifier. The biggest advantage of this approach is that it is a lot faster as hand detection does not have to happen. Disadvantages of this approach is that user has to place his hand only in the static box for the application to work which might be troublesome for user experience. [4]

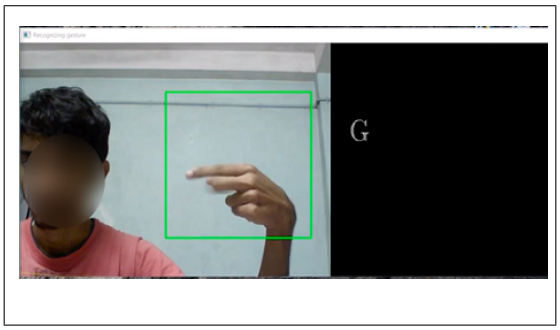


Figure 2: Static Bounding Box approach

- **Detecting Hand using Image processing and creating dynamic bounding box:** In this approach, color of the hand skin is used to detect hand position and then a bounding box is created around the center of detected hand. This approach is again very fast but this is not generalized to be used by every user. We had to always train the model using our hand under same background and lighting conditions.

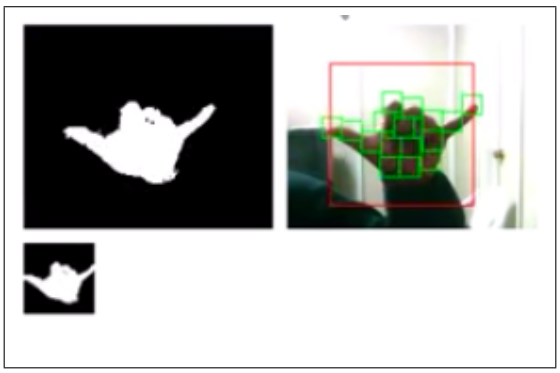


Figure 3: Detecting Hand using Image processing and creating dynamic bounding box

While Neural Networks have been applied to ASL sign recognition in the past with a good accuracy score but many of them require a 3-D capture element with motion-tracking gloves or a Microsoft Kinnect, The constraints imposed by the extra requirements reduce the scalability and feasibility of these solutions.

2 Architecture

We have developed a real time sign language classification system with client-server architecture. Our web application detects your hand from the web-cam input and classifies your hand gesture based upon our trained model.

2.1 Design

- Client Browser contacts the Web Service with a HTTP POST request containing the image.
- The Web Service has our models loaded and running.
- It passes the data received from the client to model_1 which is The Hand detection model.
- This model_1 passes a cropped image of the hand's ROI(Region of interest) to the model_2, which is our sign language recognition model.
- The model_2 returns the predicted sign letter along with its accuracy to the Web service.
- The web service returns a JSON formatted response to the client.

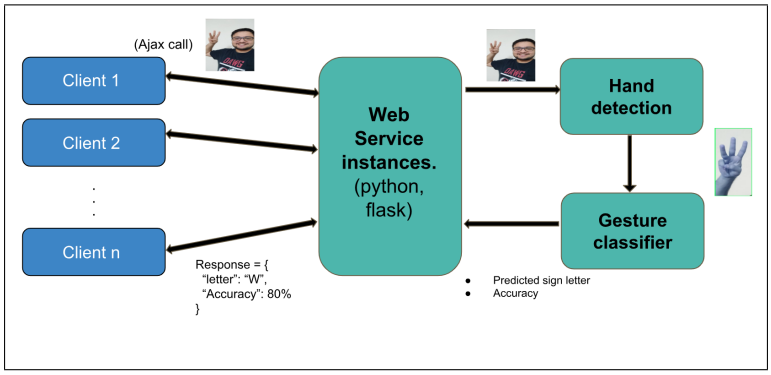


Figure 4: Overall Architecture of the application

2.2 Front-End: Web page

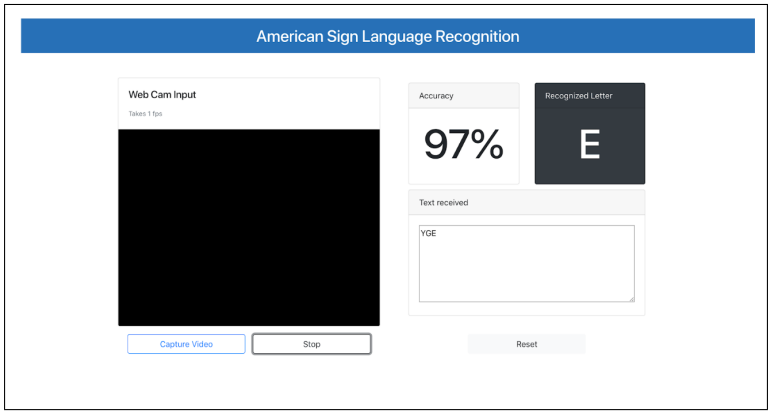


Figure 5: Front-end Design showing: web-camera input, letter predicted, accuracy score and collection of predicted letters in a text box.

We have designed a web page using which the user can interact with our gesture recognition system in real time. The Web page is designed using HTML, CSS, JavaScript, and bootstrap. It shows you the web cam input, the recognized sign letter, accuracy/confidence score, and also there is a text box where the recognized letters are appended. So basically, a sign language user can type in words and sentences using our website.

Since we have not implemented a queuing and synchronization mechanism, to avoid overload and also to get the corresponding response for the input from web cam, we have kept the sampling at 1 fps.The JavaScript code sends images clicked from the web cam at an interval of every 1 sec. over an Ajax call to the web service. The image is converted to base64 format before sending, as sending an image file over an Ajax call is not a trivial or correct solution. The JSON response received is appropriately displayed on the web page as visible in the screen-shot below.

2.3 Web Service

Often there's a need to abstract away your machine learning model details and just deploy or integrate it with easy to use API endpoints. For eg., We can provide a URL endpoint using which anyone can make a POST request and they would get a JSON response of what the model has inferred without having to worry about its technicalities. So, for a clean user interface and accessibility we decided to build a web service which would serve our models. The web service is designed in python and Flask.

The Web service is constantly running to cater to the client requests. It accepts the base64 image string and converts it to a .jpg image. This recovered image is sent to our models and a result is received. The web service formats this result into a JSON object and sends back to the client who requested it.

3 Real-time Hand-Detector: TensorFlow object detection API

The main goal of hand detection algorithm is to detect the hand/s from the input video and then create a bounding box on that hand image which to further passed to our next model. The major challenge we face here was to find a good hand data-set of various sizes and skin color. The Egohands Data-set was the perfect fit for our model. [5]

3.1 Egohands Data-set

The EgoHands data-set contains videos of complex, first-person interactions between two people. The main intention of this data-set is to enable better, data-driven approaches to understanding hands in first-person computer vision.is data-set works well for several reasons. It contains high quality, pixel level annotations (>15000 ground truth labels) where hands are located across 4800 images. All images are captured across 48 different environments (indoor, outdoor) and activities (playing cards, chess etc).While the egohands data-set provides four separate labels for hands (own left, own right, other left, and other right), for our purpose,we are only interested in the general 'hand' class and label all training data as 'hand'. [6]

3.2 Training

Transfer learning is a machine learning method where a model developed for a task is reused as the starting point for a model on a second task.It is a popular approach in deep learning where pre-trained models are used as the starting point on computer vision tasks given the vast compute and time resources required to develop neural network models on these problems and from the huge jumps in skill that they provide on related problems.[7]

As mentioned earlier, we are using TensorFlow Object Detection API. The TensorFlow Object Detection API is an open source framework built on top of TensorFlow that makes it easy to construct, train and deploy object detection models. It is based and trained on COCO API data-set. Also, TensorFlow Object Detection API uses Protobufs to configure model and training parameters. Before the framework can be used, the Protobuf libraries must be compiled. We choose `ssd_mobilenet_v1_coco` [8] as our base model. Once training is completed, we save a frozen copy of the model on our local disk so that we can use this model to detect hand from images. A frozen copy of the model is also available on Git-hub. [9]

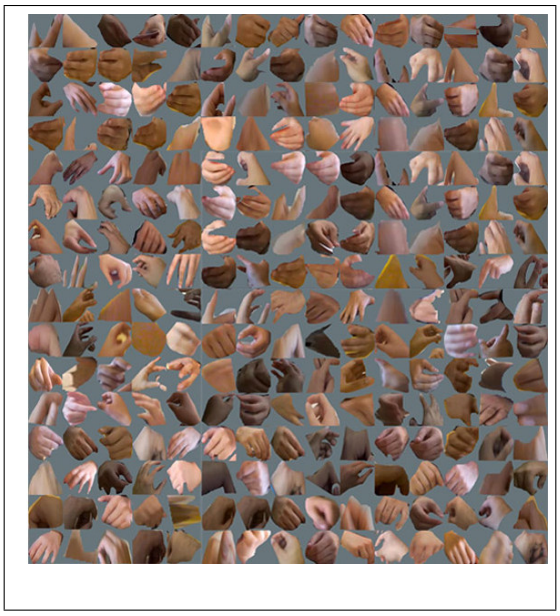


Figure 6: Sample image showing Egohands Data-set

3.3 Model Prediction

For model prediction we take the frozen model from the memory and then read the input image which is captured from a live video stream. Once the image is read we detect hand/s and visualize detected bounding boxes. This bounding box is further passed to our next model for hand gesture prediction.

4 American Sign Language Classifier

The input of this model is the output of our previous model and then it classifies the hand images into various characters and prints the accuracy score. Multiple models and data-sets were tried/tested for this application. The below mention data-set and VGG16 network worked best for our project.

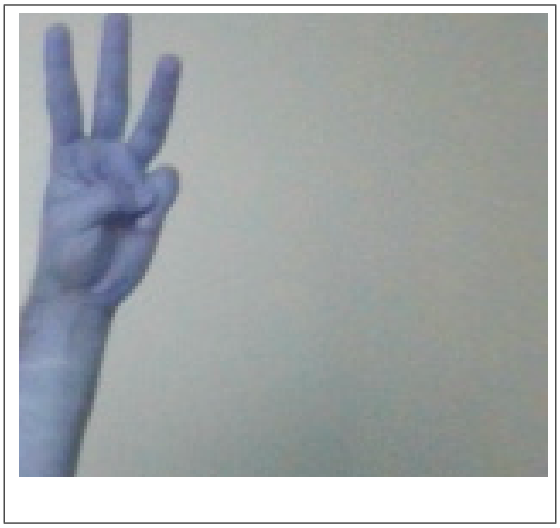


Figure 7: Detected hand (Cropped from web-cam image)

4.1 ASL Alphabet Data-set

We used ASL Alphabet data-set from Kaggle to train our classifier which is available on this website:

`https://www.kaggle.com/grassknoted/asl-alphabet/home`

This data-set is a collection of American sign language alphabets along with three signs for Space, Delete and Nothing. This amounts to 29 classes. The three additional signs indicate that this data-set was created exclusively to aid web-cam based chat applications.

In training data-set, there are total 87,000 images which all are 200x200 pixels. There are total 29 folders containing 29 class images. 26 of them are American alphabets A-Z and 3 of them are Space, Delete and Nothing.

In testing data-set, there are only 29 images to encourage us to test this data-set using real-world applications.

This data-set contains varied backgrounds and light conditions so that predictions can be made under all conditions. Additionally, these images are taken with various distances from web-cam which also aids prediction. This can be seen in figure 6.

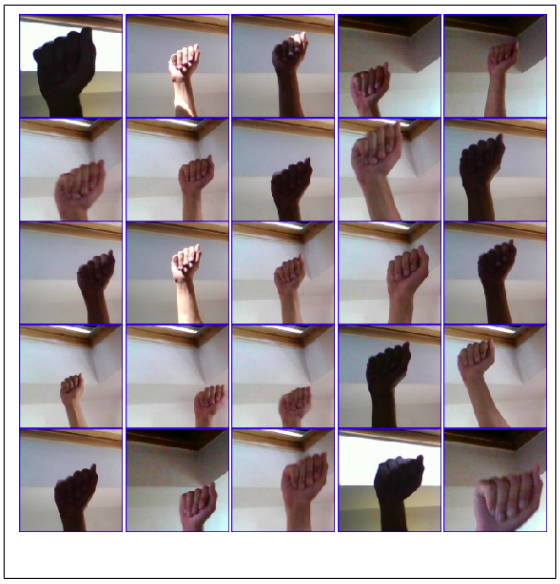


Figure 8: Various images for "A" alphabet

4.2 Pre-processing

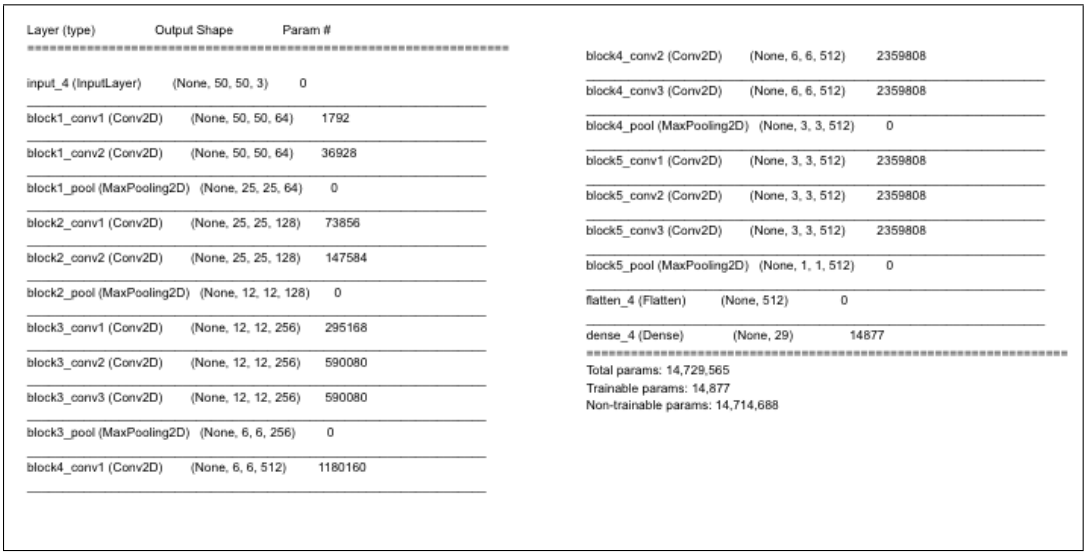
Most of the core pre-processing is done in the Hand detection where from the raw web-cam image, we detect hand and crop the bounding box. For the cropped image, we just re-size the image to 50x50 size for faster training. We also convert labels to hot vectors (ex : 2 -> [0,0,1,0,0,0,0,0,0,0]) which is supported by Keras [10].

4.3 VGG16 model

We followed below process to utilize Transfer learning as described in section 2.2:

- We used VGG16 model pre-trained on Image-Net data-set [11].
- We added a Fully connected layer at the end with only 29 nodes (Representing 29 classes of ASL Alphabet data-set).
- We then made top layers of this model untrainable because we want retain the knowledge of top layers.

Then, we get around 14,877 trainable params and 14,714,688 untrainable params. Our model summary is as shown in Figure 7:



Layer (type)	Output Shape	Param #
input_4 (InputLayer)	(None, 50, 50, 3)	0
block1_conv1 (Conv2D)	(None, 50, 50, 64)	1792
block1_conv2 (Conv2D)	(None, 50, 50, 64)	36928
block1_pool (MaxPooling2D)	(None, 25, 25, 64)	0
block2_conv1 (Conv2D)	(None, 25, 25, 128)	73856
block2_conv2 (Conv2D)	(None, 25, 25, 128)	147584
block2_pool (MaxPooling2D)	(None, 12, 12, 128)	0
block3_conv1 (Conv2D)	(None, 12, 12, 256)	295168
block3_conv2 (Conv2D)	(None, 12, 12, 256)	590080
block3_conv3 (Conv2D)	(None, 12, 12, 256)	590080
block3_pool (MaxPooling2D)	(None, 6, 6, 256)	0
block4_conv1 (Conv2D)	(None, 6, 6, 512)	1180160
block4_conv2 (Conv2D)	(None, 6, 6, 512)	2359808
block4_conv3 (Conv2D)	(None, 6, 6, 512)	2359808
block4_pool (MaxPooling2D)	(None, 3, 3, 512)	0
block5_conv1 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv2 (Conv2D)	(None, 3, 3, 512)	2359808
block5_conv3 (Conv2D)	(None, 3, 3, 512)	2359808
block5_pool (MaxPooling2D)	(None, 1, 1, 512)	0
flatten_4 (Flatten)	(None, 512)	0
dense_4 (Dense)	(None, 29)	14877

Total params: 14,729,565
Trainable params: 14,877
Non-trainable params: 14,714,688

Figure 9: Model Summary

4.4 Model Training

We used Google Cloud platform for training. We setup a Ubuntu 18.04 platform with two Nvidia Tesla K80 GPU's with 4 GB RAM each. We used CUDA [12] library to support GPU based TensorFlow framework. We used 8 CPU's with 30 GB Ram.

When we create a setup like above, it provides us CLI only meaning, it does not have any GUI so we won't be able to use jupyter notebook. We used this tutorial [13] to run jupyter notebook on our local browser using VM API network address. We used a soft-max-based loss function, It takes a feature vector z for a given training example, and squashes its values to a vector of [0,1]. Using a softmax-based classification allows us to output values akin to probabilities for each ASL letter.

$$Loss = \frac{1}{N}\sum_{i=1}^{N} -\log\left(\frac{e^{f_{i,y_i}}}{\sum_{j=1}^{C} e^{f_{i,j}}}\right)$$

$$f_j(z) = \frac{e^{z_j}}{\sum_{k=1}^{C} e^{z_k}}$$

N = total number of training examples
C = total number of classes

Figure 10: softmax-based loss function

Training took around 180 minutes to complete and we got 91.16% accuracy. Our Training Validation Accuracy vs No. of Epochs is as shown in Figure 9.

4.5 Evaluation

As mentioned before, we had only 29 images in test data set. Our model classified each of this image correctly. However, It is to be noted that, we are not going to use images exactly like training or

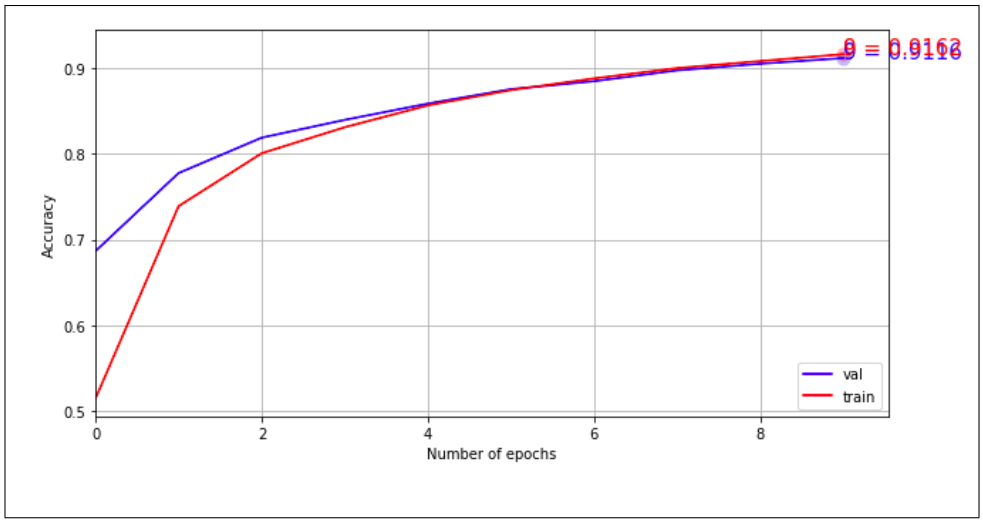


Figure 11: Training-Validation Accuracy vs No. of Epochs

testing data set. We are going to use real time images coming from web-cam which might have less resolution or different background (Can't cover all backgrounds in training data set). Even for such detected hand images, model predicts decently. We noted that especially for Sign Language alphabets like 'W', 'G', 'F', 'M' etc which are very different from each other, our model predicted accurately.

5 Challenges Overcome

- **Deployment of Tensor flow model on Web Service:** Initially we were loading the model on every call to the web service. To understand how this badly impacts the system performance, consider we have 10 concurrent users trying to use our website. Every user makes a request per second. So for the web service it becomes 10 requests per second. Now loading the machine learning models 10 times within a second puts up a lot of load on the system and our application becomes slow.

  So, we then decided to load the model only once for the web service instance and then for every request it will just predict using the loaded model. This had its own challenges with few resources on the internet to help. A web service is multithreaded since it is able to serve multiple requests simultaneously. This feature of web service created a problem for us as it started throwing multiple errors and we had no clue as to what the actual error was. We tried switching from TensorFlow back-end to Theano back-end but that did not help. Eventually we loaded default_graph object of TensorFlow and put our code inside it. This worked like a charm and we started getting our results in the front-end web page.
- **Finding a Good data-set:** As mentioned before, our aim was to classify web-cam generated and cropped image. We tried different data-sets with different models which was giving good Validation accuracy but was not correctly predicting real-time web-cam images. After some research we found the ASL Alphabet data-set which worked decently for our application
- **Modifying TensorFlow Object Detection API for Hand Detection:** The out of box TensorFlow Object Detection API works directly on web-cam image input and has multi-threaded architecture. We faced issues converting it to work on single image (frame) input passed by web service client. In the end We were able to achieve this.

6 Future Work

We plan to continue working on this project such that it can be tested and used by other users . Some of the things we plan to improve/implement are as follows:

- Improve `hand detection model` with other hands variations.

- Augment the `data-set`; Train model on more images with different hand sizes.
- Try to implement/test various other models like Inception_v3
- Ensemble and run two or more classification models to improve sign prediction accuracy.
- There is a slight delay in our client-server architecture. We will try to minimize this by deploying it on faster and powerful server as well as improving our client-server connection/handling requests.
- Long-term goal includes adding gesture based recognition to our application in order to predict words rather than predicting single letter. This can be done using Recurrent Neural Networks.

7 Conclusion

We built a Real-time American Sign Language detection application which can primarily serve as a web based chat platform. Our approach also shows that we can build a Gesture classifier which can generalize for various users and does not require user to train the model before using it. This was largely possible because of two tricks: `Hand Detection` using TensorFlow Object Detection API and utilizing massive ASL Alphabet Data-set.

Acknowledgments

This work would not have been possible without the support of Dr. Yi Hong, [Assistant Professor, Departments of Computer Science, University of Georgia] who worked actively to provide us with academic time and advice to pursue those goals. Dr. Yi Hong also provided us with Google Cloud Platform Credits which helped us to train our model more faster using superior CPU and GPU's. We would also like to thank our classmates who encouraged and motivated us by voting our project to be the best in class.

References

[1] https://www.kaggle.com/grassknoted/asl-alphabet

[2] https://www.nidcd.nih.gov/sites/default/files/Documents/health/hearing/NIDCD-American-Sign-Language.pdf

[3] http://cs231n.stanford.edu/reports/2016/pdfs/214_Report.pdf

[4] Saha, D (2018, May 9). Sign-Language (Version 1). figshare. https://doi.org/10.6084/m9.figshare.6241901.v1A very simple CNN project.

[5] http://vision.soic.indiana.edu/projects/egohands/

[6] https://towardsdatascience.com/how-to-build-a-real-time-hand-detector-using-neural-networks-ssd-on-TensorFlow-d6bac0e4b2ce

[7] https://machinelearningmastery.com/transfer-learning-for-deep-learning/

[8] https://arxiv.org/pdf/1512.02325.pdf

[9] https://github.com/victordibia/handtracking/tree/master/hand_inference_graph

[10] https://keras.io/

[11] http://www.image-net.org/

[12] https://developer.nvidia.com/cuda-zone

[13] https://towardsdatascience.com/running-jupyter-notebook-in-google-cloud-platform-in-15-min-61e16da34d52