International Journal of Research in Engineering, Science and Management
Volume 4, Issue 6, June 2021
https://www.ijresm.com | ISSN (Online): 2581-5792

Machine Learning Based Real Time Sign Language Detection

P. Rishi Sanmitra[1], V. V. Sai Sowmya[2], K. Lalithanjana[3*]
[1,2,3]*Student, Department of Computer Science and Engineering, CMR Technical Campus, Hyderabad, India*

***Abstract*: In this paper, a real time ML based system was built for the Sign Language Detection using images that have been captured with the help of a PC camera. The main purpose of this project is to design a system for the differently abled people to communicate with others with ease. This model is one of the first models to detect signs irrespective of their sign language standards (i.e., the American Standard or the Indian Standard). The existing digital translators are very slow since every alphabet has to be gestured out and the amount of time it would take to just form a simple sentence would be a lot. This model, which was trained using the SSD ML Algorithm, overcomes the above problem by directly recognizing the signs as words instead of alphabets. This model was trained using a set of 20 images for a particular sign in different conditions such as different lighting, different skin tones, backgrounds, etc., in order to increase the accuracy of detecting the gesture. The system displayed a high accuracy for all the datasets when new test data, which had not been used in the training, were introduced. The results have shown a high accuracy of 85% for the sign detection.**

***Keywords*: Deep Learning SSD ML algorithm, LabelImg software, real time, TensorFlow object detection module.**

1. Introduction

A very few people know how to communicate using sign language as it is not a mandatory language to learn. This makes it difficult for differently abled people to communicate with others.

The most common means to communicate with them is with the help of human interpreters, which is again very expensive and not many can afford it. There are many different sign languages in the world. There are around 200 sign languages in the world including Chinese, Spanish, Irish, American Sign Language and Indian Sign Language, which are the most commonly used sign languages.

The ML based Sign Language Detection system aims at communicating with differently abled people without the help of any expensive human interpreter. This model translates the signs/gestures captured into text so that the user can simply read and know what the person is trying to convey irrespective of whether the user has knowledge about the sign language or not.

2. Literature Survey

The first approach in relation to sign language recognition was by Bergh in 2011 [1]. Haar wavelets and database searching were employed to build a hand gesture recognition system. Although this system gives good results, it only considers six classes of gestures.

In a study by Balbin et al. [2], the system only recognized five Filipino words and used colored gloves for hand position recognition; our model can be trained for different gestures and can be recognized without any colored gloves and using only bare hands.

In our model the images are captured using a PC Cam and were able to get an accuracy of 75% at an average. In other models these were captured using motion sensors, such as electromyography (EMG) sensors [3], RGB cameras [4], Kinect sensors [5] and their combinations. Although the accuracy of detecting the signs is high, they also have limitations; first is their cost, as they require large-size datasets with diverse sign motion they go toned a high-end computer with powerful specifications; whereas in our model this can be achieved with minimum specifications.

The SSD model was also adopted for hand detection. The proposed model was evaluated based on the IsoGD dataset, which achieved 4.25% accuracy [6].

In 2016, with the aim of real-time object detection in testing images, two novel algorithms came out, namely, YOLO and SSD [7], [8]. YOLO uses CNN to reduce the spatial dimension detection box. It performs a linear regression to make boundary box predictions. In the case of SSD, the size of the detecting box is usually fixed and used for simultaneous size detection. Therefore, the purported advantage of SSD is known to be the simultaneous detection of objects with various sizes.

In comparison to other systems which only recognized ASL alphabets, our model is mainly for recognizing gestures, making it more useful and effective. In the literature [9]-[12], the systems only recognized ASL alphabets.

3. Methodology

A. System Architecture

In this project, a real time sign recognition ML model was built with the help of LabelImg software and TensorFlow Object Detection API, using real coloring images. This system was divided into three main phases; Initially we wrote some code to automate the picture taking process, once the pictures were taken, we used the LabelImg software to segregate these

*Corresponding author: lalithanjanakollipara@gmail.com images into the appropriate labels. These labels are named in such a way that they express the meaning of the gesture made. Once the labelling of the images was done, we have two sets of files for each image taken, one which has the actual image in it and the other being an XML file which contains information of where the model should be looking in the image during the training process. Once these files are generated, the training process begins, where the Machine is going to use a Deep Learning SSD ML algorithm to extract features from the desired image. Finally, after the model has been trained, it allows for the Sign Language Detection part to begin. To achieve the detection, we are using the TensorFlow Object Detection API where the extracted features from the images taken are passed onto the TensorFlow module which is going to make comparisons with the real time video present in the frame. On detection of any of these features it is going to generate a bounding box around the gesture and make the prediction. The prediction is going to be the same as the label of the image, hence it is very important to understand the gesture made so as to name the label correctly, a wrongly named label could result in a wrong prediction.

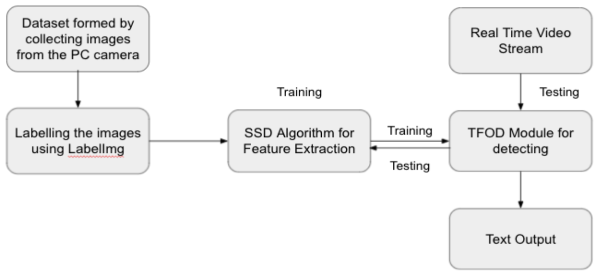

Dataset formed by collecting images from the PC camera
Labelling the images using LabelImg
Training
SSD Algorithm for Feature Extraction
Training
Testing
Real Time Video Stream
Testing
TFOD Module for detecting
Text Output

Fig. 1. System architecture

B. *Dataset Creation*

The LabelImg software is used for graphically labelling the images that is further used when recognizing the images. We have to keep in mind that labelling has to be done correctly i.e, the gesture should be labelled with a right label so that we get the gestures recognized correctly later with the right label. Once the images are labelled and saved an XML file is created for that image. This XML file contains the information about where the model should be looking in the image during the training process.

This model is trained for 5 different gestures hence 5 different labels were used for labelling them. For each gesture 20 images were used that are clicked in different angles. A code is used to take the images automatically and save them in a particular folder.

The labelling is done by drawing a box around the gesture made. This box is called the Ground Truth which means a set of measurements that is known to be much more accurate than measurements from the system you are testing. The below figure (i.e., Fig. 2) demonstrates how the images are labelled using LabelImg software.

The XML file associated with a labelled image showing where the model has to search for the gesture while training the ML model is shown in the below figure (i.e., Fig. 3).

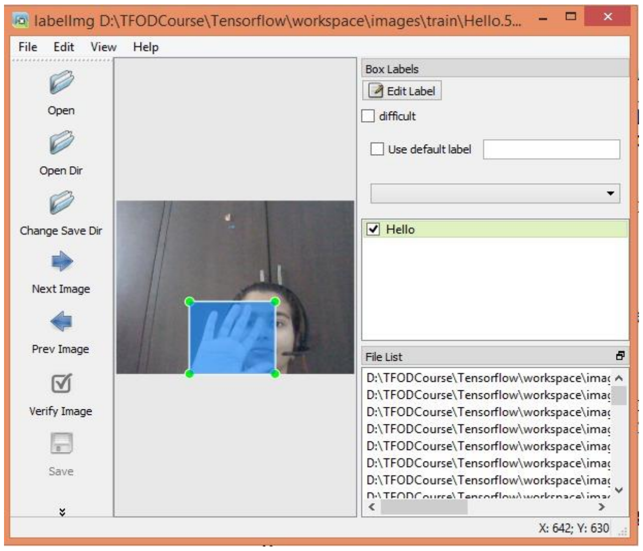
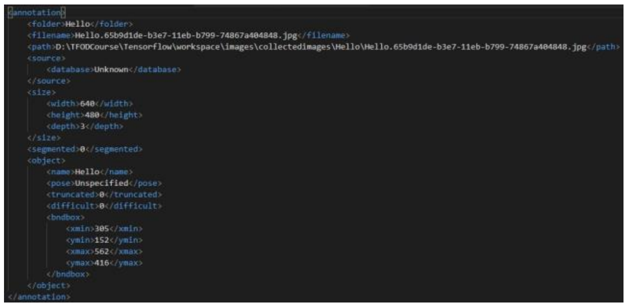


Fig. 2. Labelling the gestures using LabelImg



```
<annotation>
    <folder>Hello</folder>
    <filename>Hello.65b9d1de-b3e7-11eb-b799-74867a404848.jpg</filename>
    <path>D:\TFODCourse\Tensorflow\workspace\images\collectedimages\Hello\Hello.65b9d1de-b3e7-11eb-b799-74867a404848.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>640</width>
        <height>480</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>Hello</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>305</xmin>
            <ymin>152</ymin>
            <xmax>562</xmax>
            <ymax>416</ymax>
        </bndbox>
    </object>
</annotation>
```

Fig. 3. XML file of a labelled image

C. *Training and Testing*

Out of the 20 images collected along with the generated XML files for each image, 5 were used for testing and the remaining 15 were used for training the model. The ML model was trained using the Deep Learning SSD ML Algorithm and tested using the TensorFlow Object Detection API.

SSD (Single Shot Detection) algorithm is designed for object detection in real-time. Faster R-CNN uses a region proposal network to create boundary boxes and utilizes those boxes to classify objects. While it is considered state-of-the-art in accuracy, the whole process runs at 7 frames per second. Far below what real-time processing needs. SSD speeds up the process by eliminating the need for the region proposal network. To recover the drop in accuracy, SSD applies a few improvements including multi-scale features and default boxes. These improvements allow SSD to match the Faster R-CNN's accuracy using lower resolution images, which further pushes the speed higher.

The SSD architecture is a single convolution network that learns to predict bounding box locations and classify these locations in one pass. Hence, SSD can be trained end-to-end. The SSD network consists of base architecture (MobileNet in this case) followed by several convolution layers.

TensorFlow is an open-source library for numerical computation and large-scale machine learning that eases Google Brain TensorFlow, the process of acquiring data, training models, serving predictions, and refining future results.

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. There are already pre-trained models in their framework which are referred to as Model Zoo. It includes a collection of pre-trained models trained on various datasets such as the COCO (Common Objects in Context) dataset, the KITTI dataset, and the Open Images Dataset. The TensorFlow object detection API is the framework for creating a deep learning network that solves object detection problems.

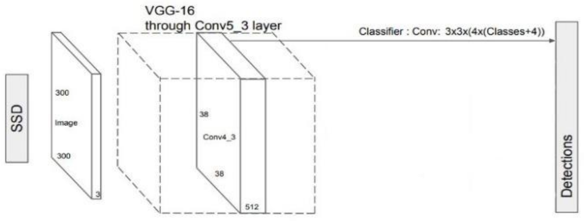


Fig. 4. SSD network architecture

TensorFlow bundles together Machine Learning and Deep Learning models and algorithms. It uses Python as a convenient front-end and runs it efficiently in optimized C++. TensorFlow allows developers to create a graph of computations to perform. Each node in the graph represents a mathematical operation and each connection represents data. Hence, instead of dealing with low-details like figuring out proper ways to hitch the output of one function to the input of another, the developer can focus on the overall logic of the application.

We use 'Checkpoints' that are save points which a model generates to keep track of how much it has trained itself. In case the training process is interrupted, it would simply start itself again from the checkpoint. Since the training process can be very time consuming, this mechanism allows the model to save itself from system failures. The learning rate of our model when used 10000 steps for training is shown below in Fig. 5.

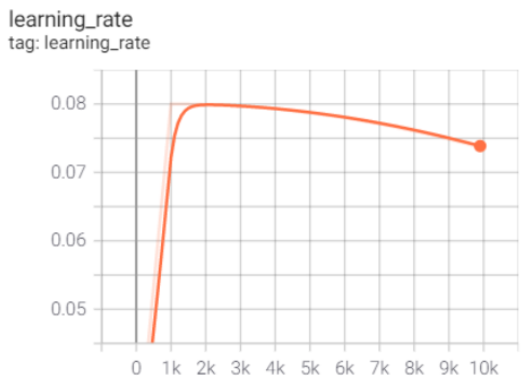


Fig. 5. Learning rate of 10000 steps training model

A loss function is used to optimize the machine learning algorithm. The loss is calculated on training and testing, and its interpretation is based on how well the model is doing in these two sets. It is the sum of errors made for each example in training or testing sets. Loss value implies how poorly or well a model behaves after each iteration of optimization. The loss at each iteration of our machine learning model has been decreasing which indicates a better accuracy of model for detection. The loss of our model is shown in the below Fig. 6.

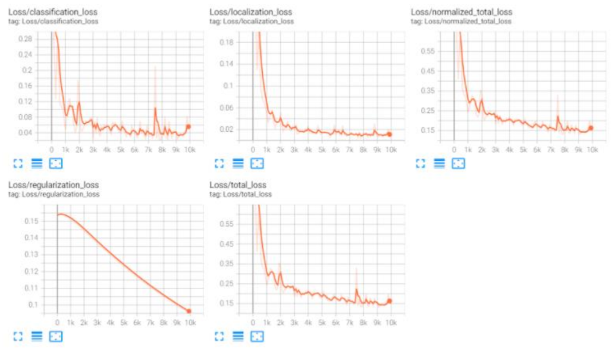


Fig. 6. Loss of the Machine Learning model

The localization loss is the mismatch between the ground truth box and the predicted boundary box. SSD only penalizes predictions from positive matches. Only the predictions from the positive matches to get closer to the ground truth is required. Negative matches can be ignored. Ground truth box is the box that is created in the LabelImg software while creating the labels and the predicted boundary box is the box that is predicted by the model while testing the images. The localization loss for our model is 0.05 as shown in Fig. 9.

The localization loss between the predicted box l and the ground truth box g is defined as the smooth L1 loss with cx, cy as the offset to the default bounding box d of width w and height h.

$$L_{loc}(x,l,g) = \sum_{i \in Pos}^{N} \sum_{m \in \{cx,cy,w,h\}} x_{ij}^{k} \text{smooth}_{\text{L1}}(l_i^m - \hat{g}_j^m)$$

$$\hat{g}_j^{cx} = (g_j^{cx} - d_i^{cx})/d_i^w \qquad \hat{g}_j^{cy} = (g_j^{cy} - d_i^{cy})/d_i^h$$

$$\hat{g}_j^{w} = \log\left(\frac{g_j^w}{d_i^w}\right) \qquad \hat{g}_j^{h} = \log\left(\frac{g_j^h}{d_i^h}\right)$$

$$x_{ij}^{p} = \begin{cases} 1 & \text{if } IoU > 0.5 \text{ between default box } i \text{ and ground true box } j \text{ on class } p \\ 0 & \text{otherwise} \end{cases}$$

Fig. 7. Formula for calculating localization loss

The confidence loss is the loss of making a class prediction. For every positive match prediction, the loss is penalized according to the confidence score of the corresponding class. For negative match predictions, the loss is penalized according to the confidence score of the class "0": class "0" classifies no object is detected. The confidence loss for our model is 0.19 as shown in Fig. 9.

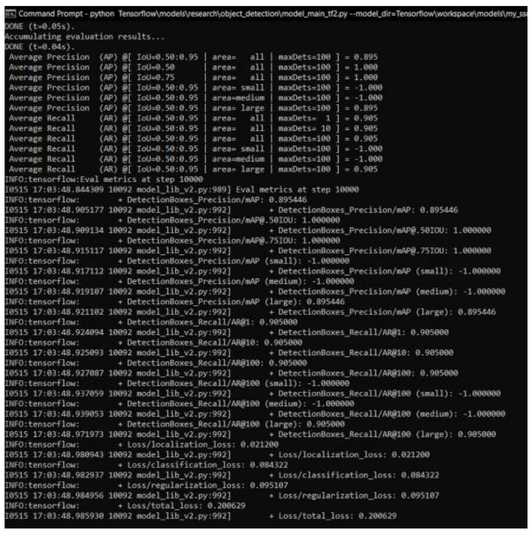
It is calculated as the softmax loss over multiple classes confidences c (class score).

$$L_{conf}(x,c) = -\sum_{i \in Pos}^{N} x_{ij}^{p} log(\hat{c}_i^p) - \sum_{i \in Neg} log(\hat{c}_i^0) \quad \text{where} \quad \hat{c}_i^p = \frac{\exp(c_i^p)}{\sum_p \exp(c_i^p)}$$

where N is the number of matched default boxes.

Fig. 8. Formula for calculating confidence loss

The below image (i.e., Fig. 9.) represents the evaluation results and evaluation metrics for a 10000-step machine learning model. An evaluation metric consists of the average precision and average recall. For each precision and recall an IOU is calculated. IOU stands for Intersection Over Union which determines the ratio of area of intersection between the Ground Truth and predicted box to the area of union between the Ground Truth and Predicted box (as shown in Fig. 10)



```
Command Prompt - python  Tensorflow\models\research\object_detection\model_main_tf2.py --model_dir=Tensorflow\workspace\models\my_ss
DONE (t=0.05s).
Accumulating evaluation results...
DONE (t=0.04s).
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.895
 Average Precision  (AP) @[ IoU=0.50      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.75      | area=   all | maxDets=100 ] = 1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Precision  (AP) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.895
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=  1 ] = 0.905
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets= 10 ] = 0.905
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=   all | maxDets=100 ] = 0.905
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= small | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area=medium | maxDets=100 ] = -1.000
 Average Recall     (AR) @[ IoU=0.50:0.95 | area= large | maxDets=100 ] = 0.905
INFO:tensorflow:Eval metrics at step 10000
I0515 17:03:48.844309 10092 model_lib_v2.py:989] Eval metrics at step 10000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP: 0.895446
I0515 17:03:48.905177 10092 model_lib_v2.py:992]        + DetectionBoxes_Precision/mAP: 0.895446
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
I0515 17:03:48.909134 10092 model_lib_v2.py:992]        + DetectionBoxes_Precision/mAP@.50IOU: 1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
I0515 17:03:48.915117 10092 model_lib_v2.py:992]        + DetectionBoxes_Precision/mAP@.75IOU: 1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (small): -1.000000
I0515 17:03:48.917112 10092 model_lib_v2.py:992]        + DetectionBoxes_Precision/mAP (small): -1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (medium): -1.000000
I0515 17:03:48.919107 10092 model_lib_v2.py:992]        + DetectionBoxes_Precision/mAP (medium): -1.000000
INFO:tensorflow:        + DetectionBoxes_Precision/mAP (large): 0.895446
I0515 17:03:48.921102 10092 model_lib_v2.py:992]        + DetectionBoxes_Precision/mAP (large): 0.895446
INFO:tensorflow:        + DetectionBoxes_Recall/AR@1: 0.905000
I0515 17:03:48.924094 10092 model_lib_v2.py:992]        + DetectionBoxes_Recall/AR@1: 0.905000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@10: 0.905000
I0515 17:03:48.925093 10092 model_lib_v2.py:992]        + DetectionBoxes_Recall/AR@10: 0.905000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100: 0.905000
I0515 17:03:48.927087 10092 model_lib_v2.py:992]        + DetectionBoxes_Recall/AR@100: 0.905000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (small): -1.000000
I0515 17:03:48.937059 10092 model_lib_v2.py:992]        + DetectionBoxes_Recall/AR@100 (small): -1.000000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (medium): -1.000000
I0515 17:03:48.939053 10092 model_lib_v2.py:992]        + DetectionBoxes_Recall/AR@100 (medium): -1.000000
INFO:tensorflow:        + DetectionBoxes_Recall/AR@100 (large): 0.905000
I0515 17:03:48.971973 10092 model_lib_v2.py:992]        + DetectionBoxes_Recall/AR@100 (large): 0.905000
INFO:tensorflow:        + Loss/localization_loss: 0.021200
I0515 17:03:48.980943 10092 model_lib_v2.py:992]        + Loss/localization_loss: 0.021200
INFO:tensorflow:        + Loss/classification_loss: 0.084322
I0515 17:03:48.982937 10092 model_lib_v2.py:992]        + Loss/classification_loss: 0.084322
INFO:tensorflow:        + Loss/regularization_loss: 0.095107
I0515 17:03:48.984956 10092 model_lib_v2.py:992]        + Loss/regularization_loss: 0.095107
INFO:tensorflow:        + Loss/total_loss: 0.200629
I0515 17:03:48.985930 10092 model_lib_v2.py:992]        + Loss/total_loss: 0.200629
```

Fig. 9. Evaluation results and evaluation metrics

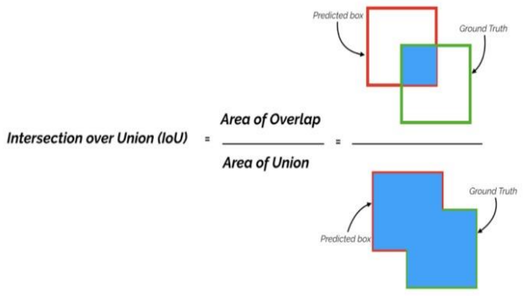
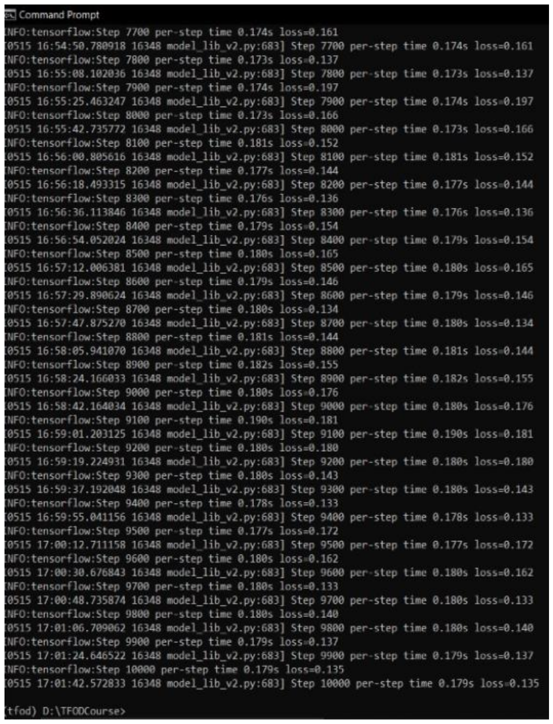


Fig. 10. Intersection over Union (IoU)



```
Command Prompt
INFO:tensorflow:Step 7700 per-step time 0.174s loss=0.161
I0515 16:54:50.780918 16348 model_lib_v2.py:683] Step 7700 per-step time 0.174s loss=0.161
INFO:tensorflow:Step 7800 per-step time 0.173s loss=0.137
I0515 16:55:08.102036 16348 model_lib_v2.py:683] Step 7800 per-step time 0.173s loss=0.137
INFO:tensorflow:Step 7900 per-step time 0.174s loss=0.197
I0515 16:55:25.463247 16348 model_lib_v2.py:683] Step 7900 per-step time 0.174s loss=0.197
INFO:tensorflow:Step 8000 per-step time 0.173s loss=0.166
I0515 16:55:42.735772 16348 model_lib_v2.py:683] Step 8000 per-step time 0.173s loss=0.166
INFO:tensorflow:Step 8100 per-step time 0.181s loss=0.152
I0515 16:56:00.805616 16348 model_lib_v2.py:683] Step 8100 per-step time 0.181s loss=0.152
INFO:tensorflow:Step 8200 per-step time 0.177s loss=0.144
I0515 16:56:18.493315 16348 model_lib_v2.py:683] Step 8200 per-step time 0.177s loss=0.144
INFO:tensorflow:Step 8300 per-step time 0.176s loss=0.136
I0515 16:56:36.113846 16348 model_lib_v2.py:683] Step 8300 per-step time 0.176s loss=0.136
INFO:tensorflow:Step 8400 per-step time 0.179s loss=0.154
I0515 16:56:54.052024 16348 model_lib_v2.py:683] Step 8400 per-step time 0.179s loss=0.154
INFO:tensorflow:Step 8500 per-step time 0.180s loss=0.165
I0515 16:57:12.006381 16348 model_lib_v2.py:683] Step 8500 per-step time 0.180s loss=0.165
INFO:tensorflow:Step 8600 per-step time 0.179s loss=0.146
I0515 16:57:29.890624 16348 model_lib_v2.py:683] Step 8600 per-step time 0.179s loss=0.146
INFO:tensorflow:Step 8700 per-step time 0.180s loss=0.134
I0515 16:57:47.875270 16348 model_lib_v2.py:683] Step 8700 per-step time 0.180s loss=0.134
INFO:tensorflow:Step 8800 per-step time 0.181s loss=0.144
I0515 16:58:05.941070 16348 model_lib_v2.py:683] Step 8800 per-step time 0.181s loss=0.144
INFO:tensorflow:Step 8900 per-step time 0.182s loss=0.155
I0515 16:58:24.166033 16348 model_lib_v2.py:683] Step 8900 per-step time 0.182s loss=0.155
INFO:tensorflow:Step 9000 per-step time 0.180s loss=0.176
I0515 16:58:42.164034 16348 model_lib_v2.py:683] Step 9000 per-step time 0.180s loss=0.176
INFO:tensorflow:Step 9100 per-step time 0.190s loss=0.181
I0515 16:59:01.203125 16348 model_lib_v2.py:683] Step 9100 per-step time 0.190s loss=0.181
INFO:tensorflow:Step 9200 per-step time 0.180s loss=0.180
I0515 16:59:19.224931 16348 model_lib_v2.py:683] Step 9200 per-step time 0.180s loss=0.180
INFO:tensorflow:Step 9300 per-step time 0.180s loss=0.143
I0515 16:59:37.192048 16348 model_lib_v2.py:683] Step 9300 per-step time 0.180s loss=0.143
INFO:tensorflow:Step 9400 per-step time 0.178s loss=0.133
I0515 16:59:55.041156 16348 model_lib_v2.py:683] Step 9400 per-step time 0.178s loss=0.133
INFO:tensorflow:Step 9500 per-step time 0.177s loss=0.172
I0515 17:00:12.711158 16348 model_lib_v2.py:683] Step 9500 per-step time 0.177s loss=0.172
INFO:tensorflow:Step 9600 per-step time 0.180s loss=0.162
I0515 17:00:30.676843 16348 model_lib_v2.py:683] Step 9600 per-step time 0.180s loss=0.162
INFO:tensorflow:Step 9700 per-step time 0.180s loss=0.133
I0515 17:00:48.735874 16348 model_lib_v2.py:683] Step 9700 per-step time 0.180s loss=0.133
INFO:tensorflow:Step 9800 per-step time 0.180s loss=0.140
I0515 17:01:06.709062 16348 model_lib_v2.py:683] Step 9800 per-step time 0.180s loss=0.140
INFO:tensorflow:Step 9900 per-step time 0.179s loss=0.137
I0515 17:01:24.646522 16348 model_lib_v2.py:683] Step 9900 per-step time 0.179s loss=0.137
INFO:tensorflow:Step 10000 per-step time 0.179s loss=0.135
I0515 17:01:42.572833 16348 model_lib_v2.py:683] Step 10000 per-step time 0.179s loss=0.135

(tfod) D:\TFODCourse>
```

Fig. 11. Loss at each iteration

A loss function is a mathematical formula used to produce loss values during training time. During training, the performance of a model is measured by the loss (L) that the model produces for each sample or batch of samples. The loss essentially measures how “far” the predicted values (y) are from the expected value (y). If y is far away (very different) from y, then the loss will be high. However, if y is close to y then the loss is low. The model uses the loss as an “indicator” to update its parameters so that it can produce very small losses in future predictions. That means producing y that are very close to y.

The figure (i.e., Fig. 11.) shows the loss incurred at each step while training the model. The lowest loss recorded is 0.133 at step 9400 and at step 9700.

4. Results and Discussions

A real-time Sign Language Detection with a SSD algorithm using real coloring images from a PC camera was introduced. In this paper, signs are translated into text statements to help the differently abled people to communicate with others with ease. This system showed good results by taking advantage of deep learning techniques. This section discusses the results obtained by the system.

Fig. 12. shows an accuracy of 90% for the recognition of the sign ‘No’ by the system.

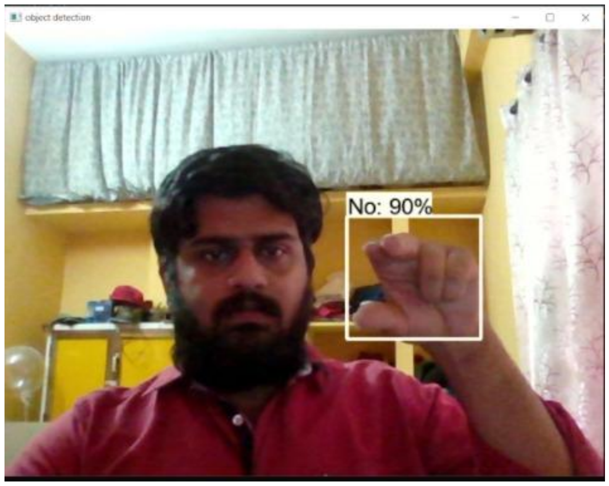


Fig. 12. Gesture recognition for No

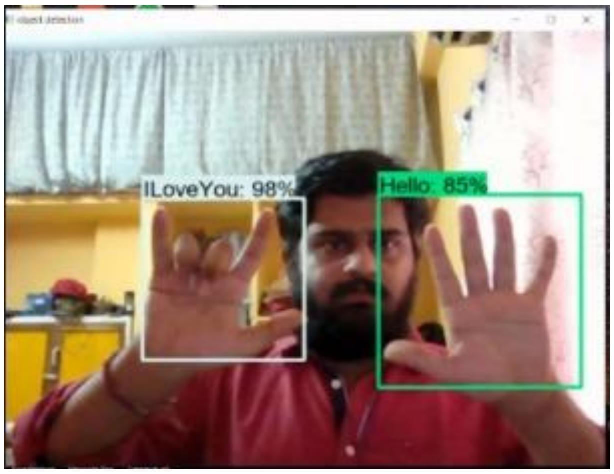


Fig. 13. Gesture recognition for ILoveYou and Hello

Two gestures can also be recognized simultaneously using our system. The accuracy has shown to be the same irrespective of two gestures being made simultaneously. In Fig. 13. we can see that both the gestures are being recognized without any difficulty.

Apart from the gestures recognized above, there are two more gestures that we used for training the model. Total five gestures were used to train the machine learning model by taking 20 images for each model in different angles, backgrounds, skin tones, lighting and other various situations. Out of the 20 images collected, 15 were used for training and 5 were for testing. All the images were converted into gray scale images as shown in Fig. 14., for training the ML model. The results have shown up to an average accuracy of 85%.

Fig. 14. Images for training in grayscale

5. Conclusion

In this paper, a real-time ML based Sign Language Recognition system was built using real coloring images that were taken with the help of a PC camera. New datasets were built to contain a wider variety of features for example different lightings, different skin tones, different backgrounds, and a wide variety of hand gestures. The system achieved a maximal accuracy of about 75% for training and 85% for the validation set. In addition, the system showed a high accuracy with the introduction of new test data that had not been used in the training. There is a lot of scope for this project, since we have the ability to label these images, we can label the gestures in any language required, allowing the user to communicate with others irrespective of the language boundaries.

References

[1] M. Van den Bergh and L. Van Gool, "Combining RGB and ToF cameras for real-time 3D hand gesture interaction," *2011 IEEE Workshop on Applications of Computer Vision (WACV)*, 2011, pp. 66-72.

[2] J. R. Balbin et al., "Sign language word translator using Neural Networks for the Aurally Impaired as a tool for communication," *2016 6th IEEE International Conference on Control System, Computing and Engineering (ICCSCE)*, 2016, pp. 425-429.

[3] J. Wu, Z. Tian, L. Sun, L. Estevez and R. Jafari, "Real-time American Sign Language Recognition using wrist-worn motion and surface EMG sensors," *2015 IEEE 12th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*, 2015, pp. 1-6.

[4] D. Mart, Sign Language Translator Using Microsoft Kinect XBOX 360 TM, 2012, pp. 1-76.

[5] Cao Dong, M. C. Leu and Z. Yin, "American Sign Language alphabet recognition using Microsoft Kinect," *2015 IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2015, pp. 44-52.

[6] Rastgoo R, Kiani K, Escalera S, "Video-based isolated hand sign language recognition using a deep cascaded model," in *Multimed. Tools Appl*. 2020, pp. 22965–22987.

[7] J. Redmon, S. Divvala, R. Girshick and A. Farhadi, "You Only Look Once: Unified, Real-Time Object Detection," *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 779-788.

[8] Liu W, Anguelov D, Erhan D, et al., "SSD: Single shot multibox detector," in *Proceedings of the European Conference on Computer Vision*," New York: Springer; 2016, pp. 21–37.

[9] T. Kim, G. Shakhnarovich, and K. Livescu, "Finger-spelling recognition with semi-markov conditional random fields," in *Proc. 2013 IEEE International Conference on Computer Vision*, 2013, pp. 1521–1528.

[10] N. Pugeault and R. Bowden, "Spelling it out: Real-time asl finger-spelling recognition," in Proc. *2011 IEEE International Conference on Computer Vision Workshop*, 2011, pp. 1114–1119

[11] S. Shahriar, A. Siddiquee, T. Islam, A. Ghosh, R. Chakraborty, A. I. Khan, C. Shahnaz, and S. A. Fattah, "Real-time american sign language recognition using skin segmentation and image category classification with convolutional neural network and deep learning," in *Proc. TENCON 2018-2018 IEEE Region 10 Conference*, 2018, pp. 1168-1171.

[12] R. Daroya, D. Peralta, and P. Naval, "Alphabet sign language image classification using deep learning," in *Proc. TENCON 2018-2018 IEEE Region 10 Conference*, 2018, pp. 646-650.