BIRLA INSTITUTE OF TECHNOLOGY
MESRA

END SEMESTER PROJECT

SIGN-TO-TEXT

Submitted By:

- **DivyaPoluri (BE/10362/17)**
- **MadhulikaPallapothula (BE/10019/17)**
- **Muskan Srivastava (BE/10011/17)**

Guided By:

Prof. Shruti Garg

7th Semester

Dept. of Computer Science and Engineering

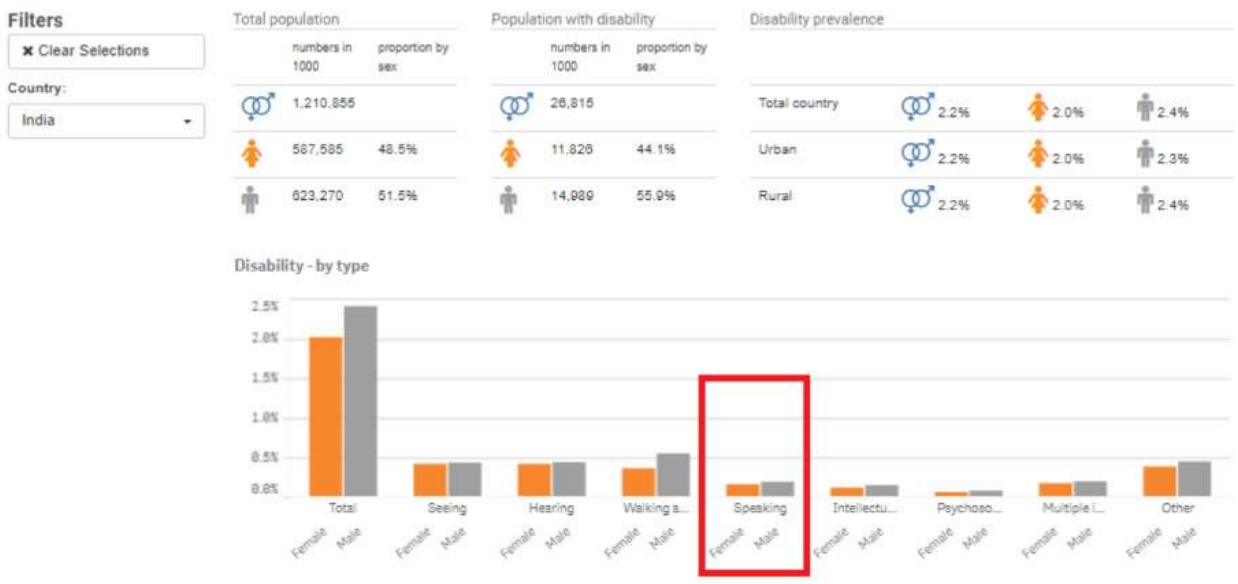
INDEX

OBJECTIVE

The goal of this project is to build a sign-language recognizer for people who cannot speak. According to the 2011 census, 7% of the population comprises of people who have speaking disabilities, and communicate using their hands. The problem, however, is that Sign Language is not very popular among the remaining 93% of the Indian population. This means that the people with speaking disabilities cannot effectively communicate face-to-face with those who can speak. Since they are most comfortable with sign language, and with so much success in machine learning, we decided to build this recognizer.

The objectives of the recognizer is to take an image as an input (or capture images from a camera), recognize a hand in the image, and classify the sign (if there is one) into 26 letters in the English alphabet. ASL is the most popular sign language in India, which is why our model is trained on ASL signs.Our dataset focuses on Indian skin tone so that the recognition of letters become more accurate.

Statistics from the United Nations:

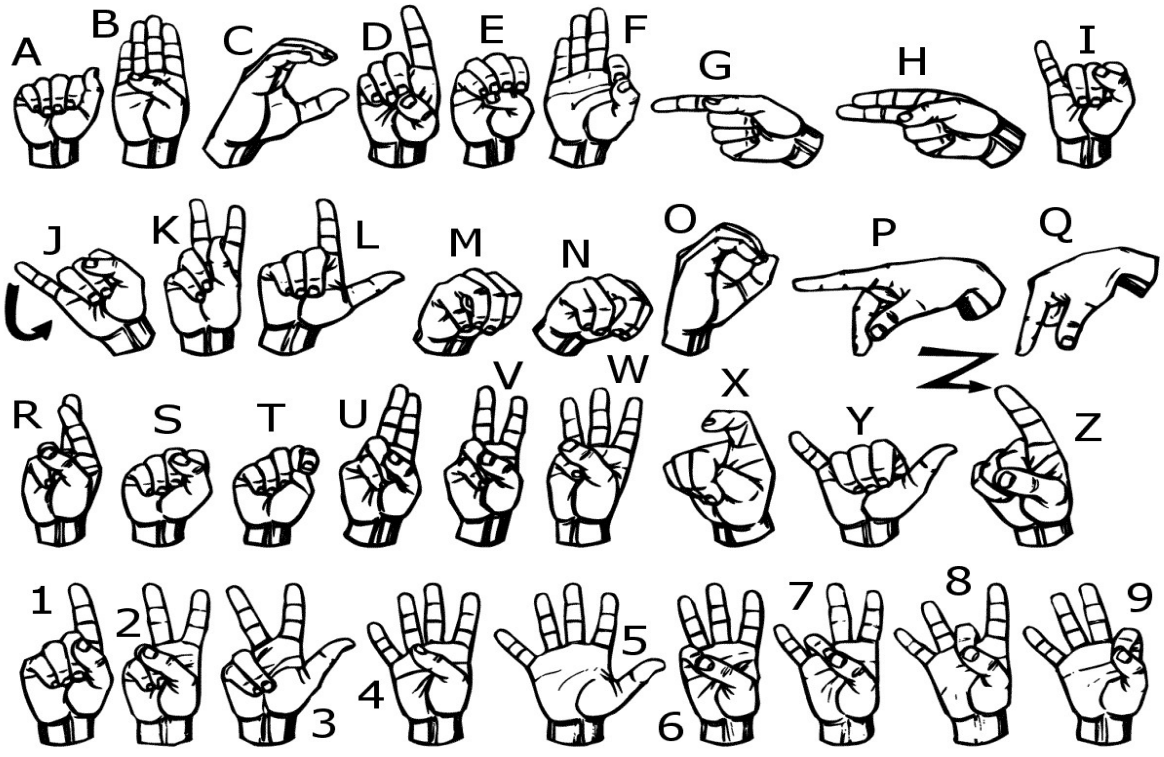


INTRODUCTION

Motivation

People with speech impairment find it very difficult to express themselves in a comprehensible way to people who do not understand sign language. They often take the help of translators and are heavily dependent on them when communicating in person with someone. Also, people who are not deaf or dumb, often do not learn sign language for interacting with those who have these disabilities. Thus, speech and hearing-impaired persons feel isolated from the rest of the world. Today there have been immense advancements in technology and in the fields of machine learning and artificial intelligence. It is quite possible to build a tool that could bridge this communication gap and give voice to people with speech and hearing impairments.

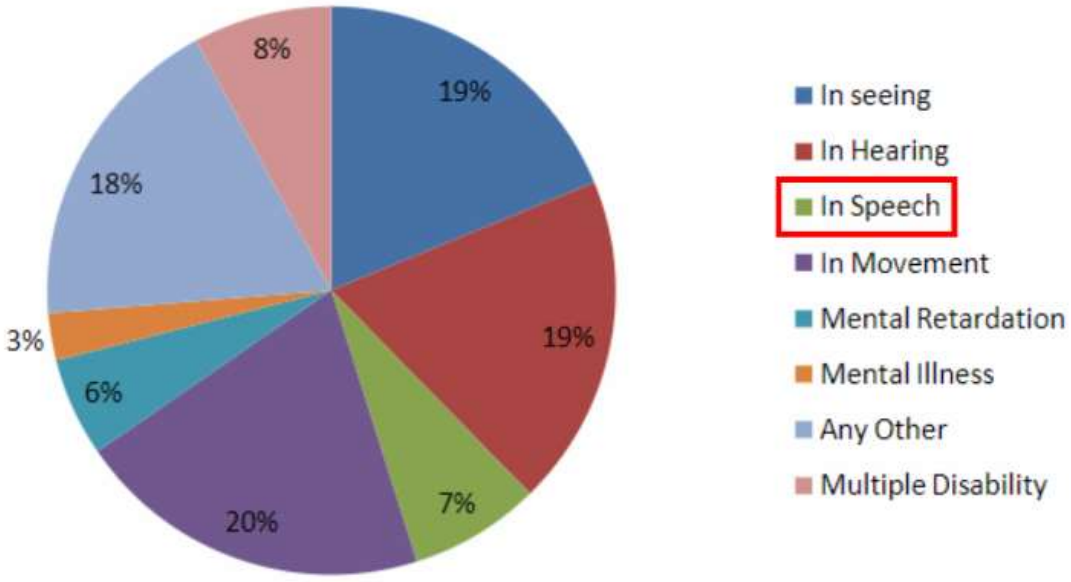
Indian Sign Language (ISL) requires both the hands for each alphabet, and is more complex. American Sign language (ASL) requires only one hand to represent each alphabet, reducing complexity. It is also more popular than ISL. Therefore, we chose ASL to build our recognizer.



As evident from the figure above, ASL has different alphabets whose signs are quite similar. For example, the letters m and n, or k and v, are hard to clearly distinguish. It is very easy to get two

completely different signs mixed up which leads to bad miscommunication. Our model also focuses on properly differentiating between such similar signs.

Distribution of disabled population by type in India (census 2011):



It is evident from the chart that 7% of the whole population has speech disability whereas 19% of them have hearing disabilities, which leads to inefficient or no communication between them and everyone else. Sign Language is the most natural and expressive way for the hearing-impaired people. People, who are not deaf, never try to learn the sign language for interacting with the deaf people. This leads to isolation of the deaf people. But if the computer can be programmed in such a way that it can translate sign language to text format, the difference between the normal people and the deaf community can be minimized.

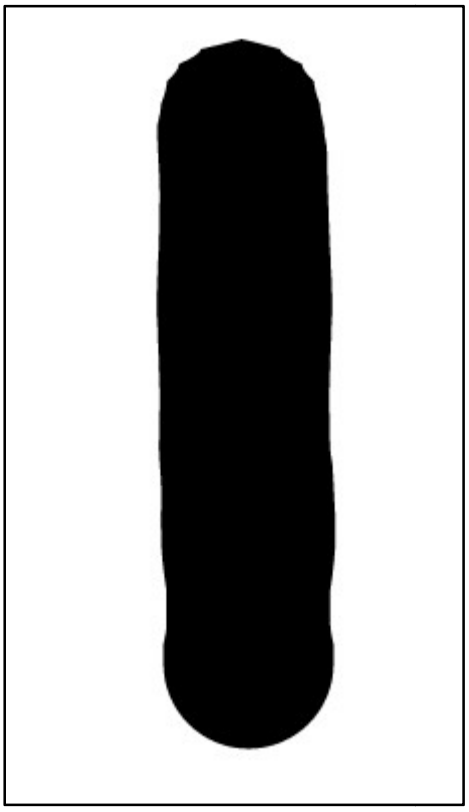
Idea

Sign Language Recognition is one of the most growing fields of research area. Many new techniques have been developed recently in this area. The Sign Language is mainly used for communication of deaf-dumb people.This project was motivated by our desire to investigate the image recognition field of machine learning since it allows to approach computer vision which is in trend.

Domain introduction

This project of recognizing sign language and converting it to text comes under image recognition in machine learning.

Images have 2 dimensions to them: height and width. These are represented by rows and columns of pixels. In this way, we can map each pixel value to a position in the image matrix (2D array so rows and columns). Machines do not really care about the dimensionality of the image; most image recognition models flatten an image matrix into one long array of pixels anyway so they don't care about the position of individual pixel values. Rather, they care about the position of pixel values relative to other pixel values. They learn to associate positions of adjacent, similar pixel values with certain outputs or membership in certain categories. Consider the example of recognizing the number 1.



And have this as the pixel values:

[[255, 255, 255, 255, 255],

[255, 255, 0, 255, 255],

[255, 255, 0, 255, 255],

[255, 255, 0, 255, 255],

[255, 255, 255, 255, 255]]

A program would not care that the 0s are in the middle of the image; it would flatten the matrix out into one long array and say that, because there are 0s in certain positions and 255s everywhere else, we are likely feeding it an image of a 1. The same can be said with colored images. If a model sees pixels representing greens and browns in similar positions, it might think it's looking at a tree (once it has been trained to look for that).

This is also how image recognition models address the problem of distinguishing between objects in an image; they can recognize the boundaries of an object in an image when they see drastically different values in adjacent pixels. A machine learning model essentially looks for patterns of pixel values that it has seen before and associates them with the same outputs. It does this during training; we feed images and the respective labels into the model and over time, it learns to associate pixel patterns with certain outputs. If a model sees many images with pixel values that denote a straight black line with white around it and is told the correct answer is a 1, it will learn to map that pattern of pixels to a 1.

This is great when dealing with nicely formatted data. If we feed a model a lot of data that looks similar then it will learn very quickly. The problem then comes when an image looks slightly different from the rest but has the same output. Consider again the image of a 1. It could be drawn at the top or bottom, left or right, or center of the image. It could have a left or right slant to it. It could look like this: 1 or this l. This is a big problem for a poorly-trained model because it will only be able to recognize nicely-formatted inputs that are all of the same basic structure but there is a lot of randomness in the world. We need to be able to take that into account so our models can perform practically well. This is why we must expose a model to as many different kinds of inputs as possible so that it learns to recognize general patterns rather than specific ones. For this reason, our recognizer was trained on over 3000 diverse images using YOLO. The Darkent53 architecture was adopted for this.

Basic idea

The basic steps involved in the project are:

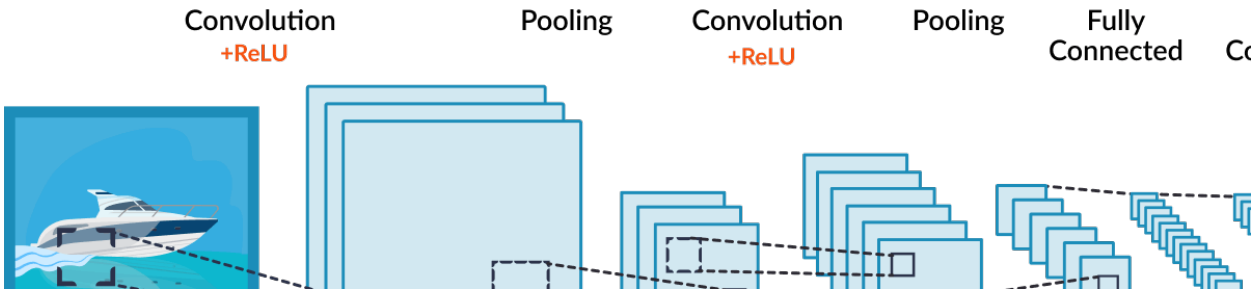
- Building a data set
- Training a machine learning model to recognise the American Sign Language (ASL).

Keywords and definitions

1. *Convolutional neural networks (CNN's)-*

A Convolutional Neural Network (CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a CNN is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, CNNs have the ability to learn these filters/characteristics.

The architecture of a CNN is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlaps to cover the entire visual area.



2. *The Kernel:*

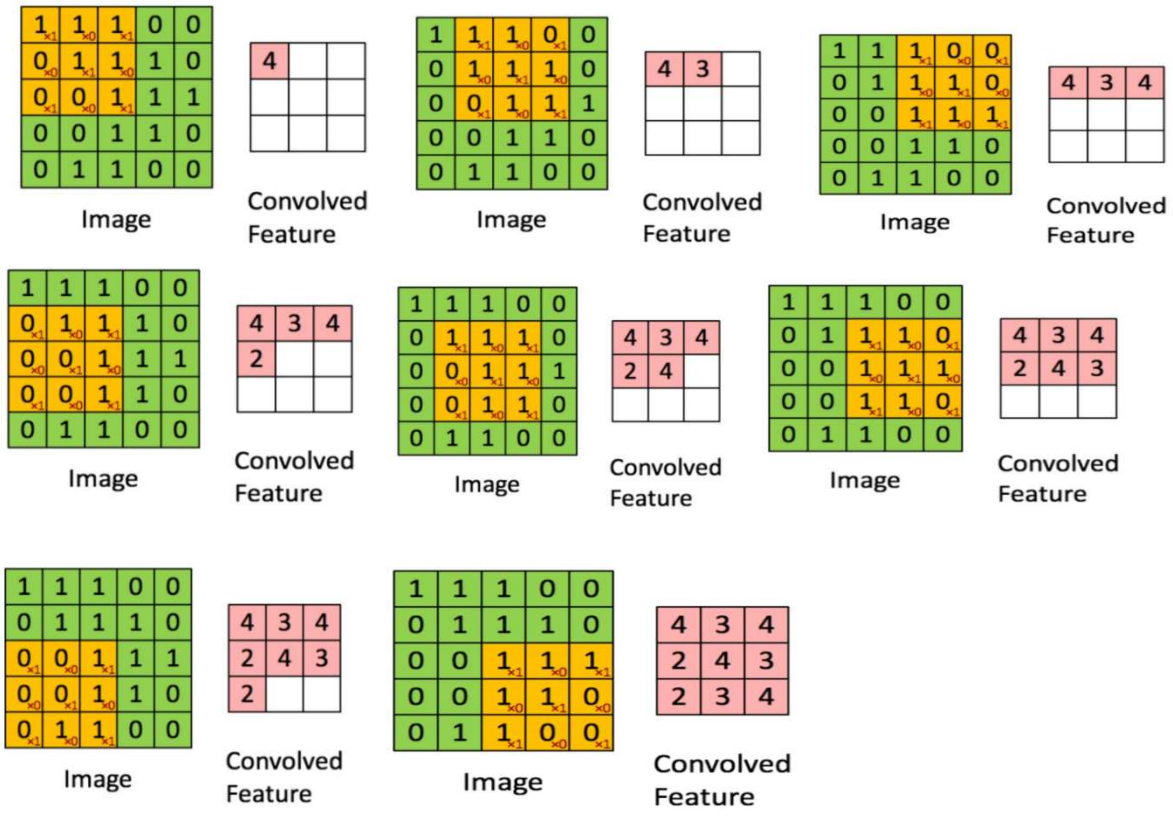


Image Dimensions = 5 (Height) x 5 (Breadth) x 1 (Number of channels, eg. RGB)

In the above demonstration, the green section resembles our **5x5x1 input image, I**. The element involved in carrying out the convolution operation in the first part of a Convolutional Layer is called the **Kernel/Filter, K**, represented in the color yellow. We have selected **K as a 3x3x1 matrix.**

Kernel/ Filter, $K = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 1 & 0 & 1 \end{pmatrix}$

The Kernel shifts 9 times because of **Stride Length = 1 (Non-Strided)**, every time performing a **matrix multiplication operation between K and the portion P of the image** over which the kernel is hovering.

The filter moves to the right with a certain Stride Value till it parses the complete width. Moving on, it hops down to the beginning (left) of the image with the same Stride Value and repeats the process until the entire image is traversed.

3. *YOLO Algorithm-*

YOLO algorithm is an algorithm based on regression, instead of selecting the interesting part of an Image, it predicts classes and bounding boxes for the whole image in one run of the Algorithm.

To understand the YOLO algorithm, first we need to understand what is actually being predicted. Ultimately, we aim to predict a class of an object and the bounding box specifying object location. Each bounding box can be described using four descriptors:

1. Centre of the box (bx, by)
2. Width (bw)
3. Height (bh)
4. Value c corresponding to the class of an object

Along with that we predict a real number pc, which is the probability that there is an object in the bounding box.

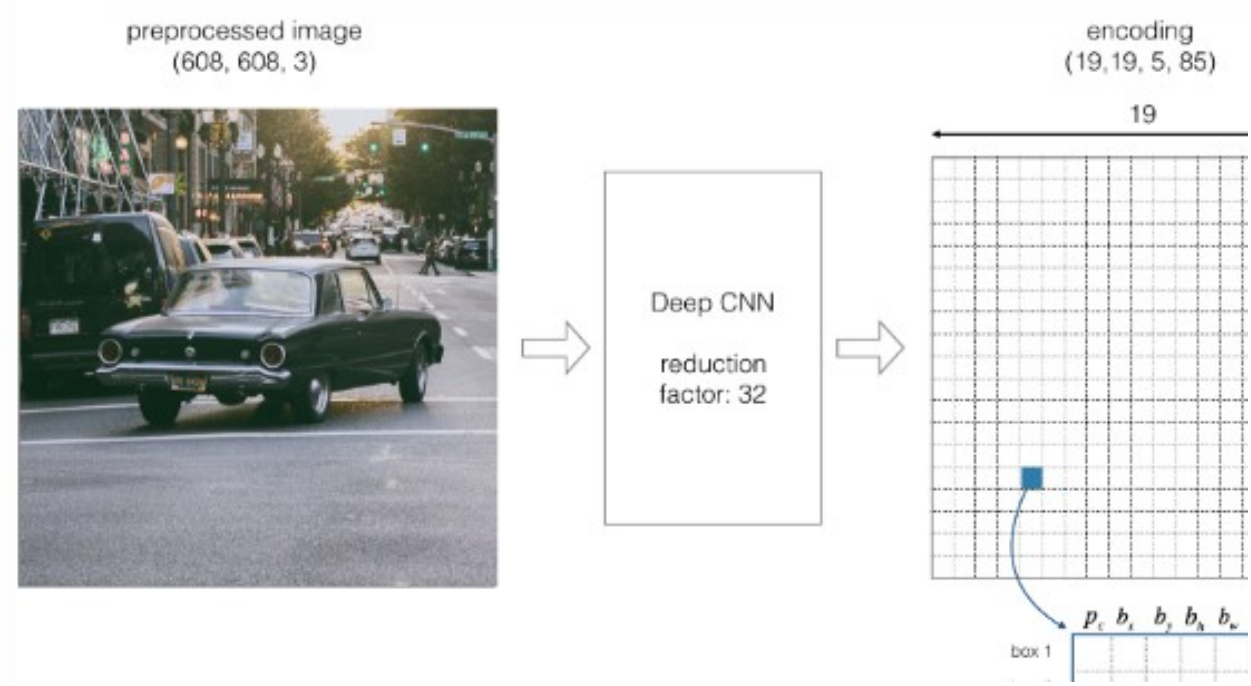
YOLO doesn't search for interested regions in the input image that could contain an object, instead it splits the image into cells, typically 19x19 grid. Each cell is then responsible for predicting K bounding boxes.

An Object is considered to lie in a specific cell only if the centre co-ordinates of the anchor box lie in that cell. Due to this property the centre co-ordinates are always calculated relative to the cell whereas the height and width are calculated relative to the whole Image size.

During the one pass of forwards propagation, YOLO determines the probability that the cell contains a certain class. The equation for the same is:

$$score_{c,i} = p_c \, x \, c_i$$

where, p_c = probability that there is an object of certain class 'c'



The class with the maximum probability is chosen and assigned to that particular grid cell. Similar process happens for all the grid cells present in the image.

After computing the above class probabilities, the image may look like this:

This shows the before and after of predicting the class probabilities for each grid cell.

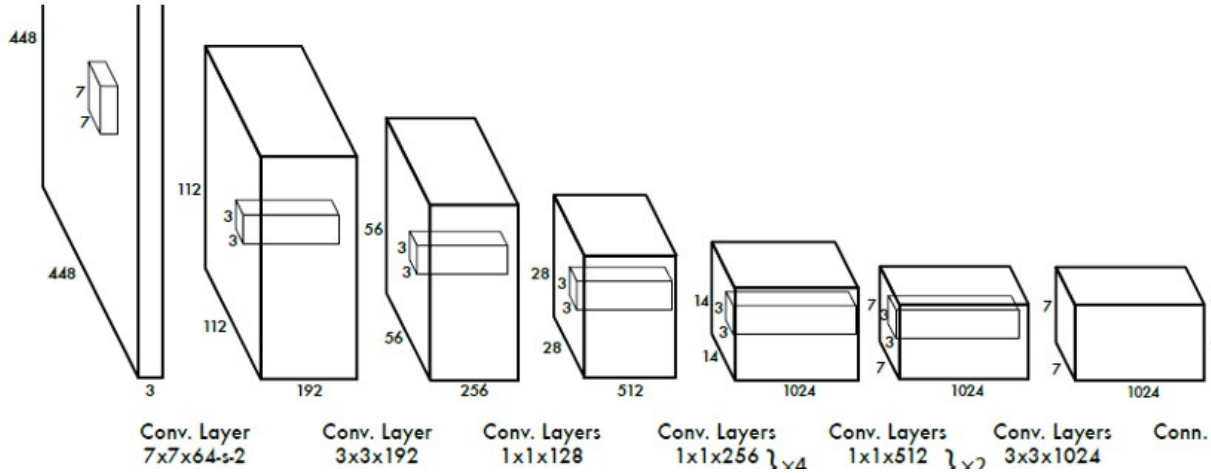


Figure: Yolo Architecture.

PROPOSED METHODS

Method-1:

This consisted of two steps:

1. Use a pre-existing data set, and train it on yolo for detecting hands and drawing bounding boxes.
2. Extract the image covered in bounding box and feed it to a simpler Neural Network similar to the AlexNet architecture that would be trained on another pre-existing dataset similar to the mnist dataset of 28x28 grayscale images.

Step 1 – Training YOLO:

The EgoHandsDataset -

There are two popular datasets of hands on the internet – the EgoHands dataset and the Oxford Hand Dataset.

The EgoHands dataset contains 48 Google Glass videos of complex, first-person interactions between two people. The main intention of this dataset is to enable better, data-driven approaches to understanding hands in first-person computer vision. The dataset offers

- high quality, pixel-level segmentations of hands
- the possibility to semantically distinguish between the observer's hands and someone else's hands, as well as left and right hands
- virtually unconstrained hand poses as actors freely engage in a set of joint activities
- lots of data with 15,053 ground-truth labelled hands.

The EgoHands dataset was more promising for our purpose and so we first trained on this.

This dataset comprises of 4,400 training images and 400 test images. These training images were trained for over 2000 iterations on YOLO.

Step 2 – Training the second Network:

Dataset

The ASL dataset on Kaggle was used to train this network. The architecture consisted of 3 convolution layers, each followed by a Max Pool layer, which was followed by 2 fully connected layers before the final layer consisting of 26 nodes for each letter.

The trained weights from YOLO would then be used to extract the hand from the picture and this cropped image would then be sent into the secondneural network and tell which sign was shown in the image.

Though this approach looked promising at first, this method did not meet our requirements because of two reasons:

1. The weights from training YOLO on the EgoHands dataset failed to properly detect 10 letters(a,d, e, g, h, k, m, n, s, t) properly.
2. Bounding boxes drawn by the weights from the YOLO trained network did not always capture the palm that was necessary for the second architecture to detect signs(example shown in figure below).

figure

Due to these reasons, a new approach was decided.

Method 2:

Since there were no existing dataset that would meet our requirement, we had to create our own dataset. Our custom dataset consists of 4,338 images which were split into 80:20 training:test data (3561 training images and 777 test images). These images were annotated using LabelImg. These annotations consist of 27 classes (0:hand, 1:A, 2:B, …, 25:Y, 26:Z). It should be noted here that the letters I and Z require motion of hands, and thus our recognizer was not trained to recognize those two letters.

Generating annotations for images:

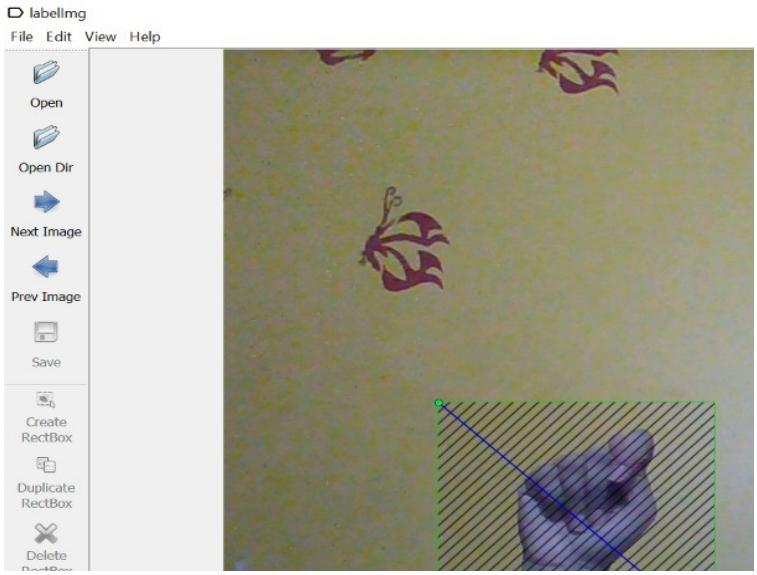


Figure 1. Creating accurate bounding box.

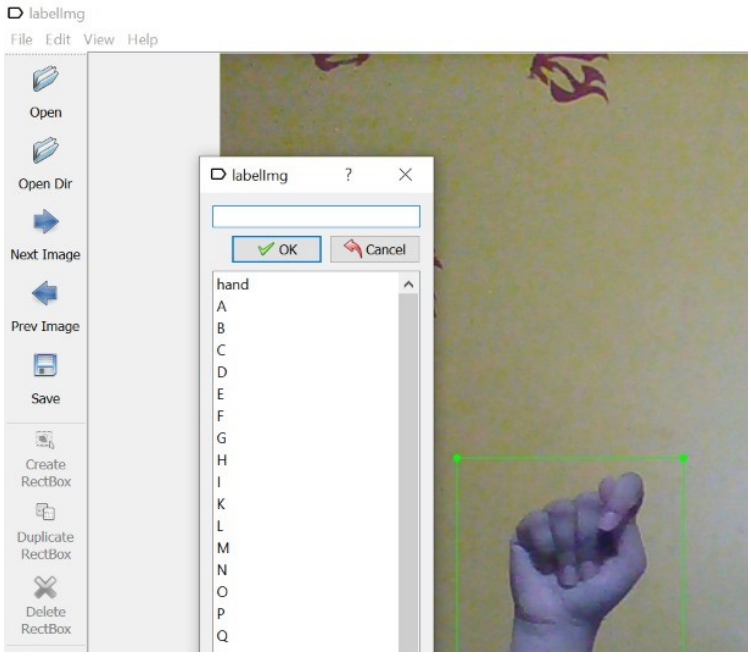


Figure 2: Labelling the image.

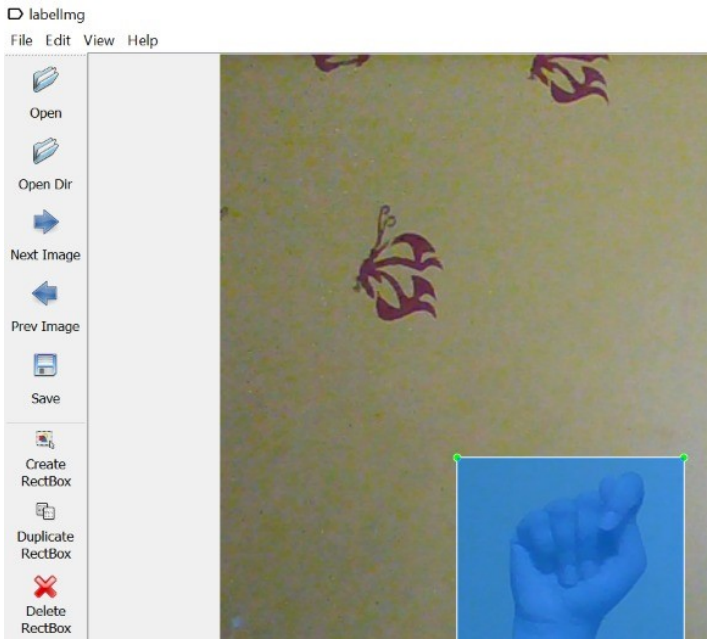


Figure 3. Labelled bounding box created.

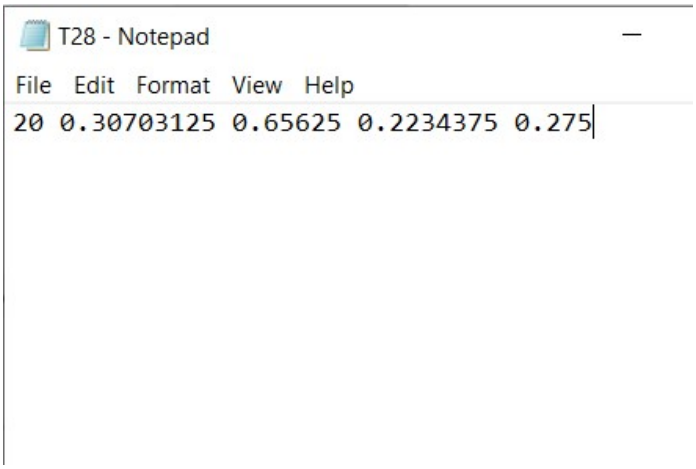


Figure 4: Text file of the bounding box was generated.

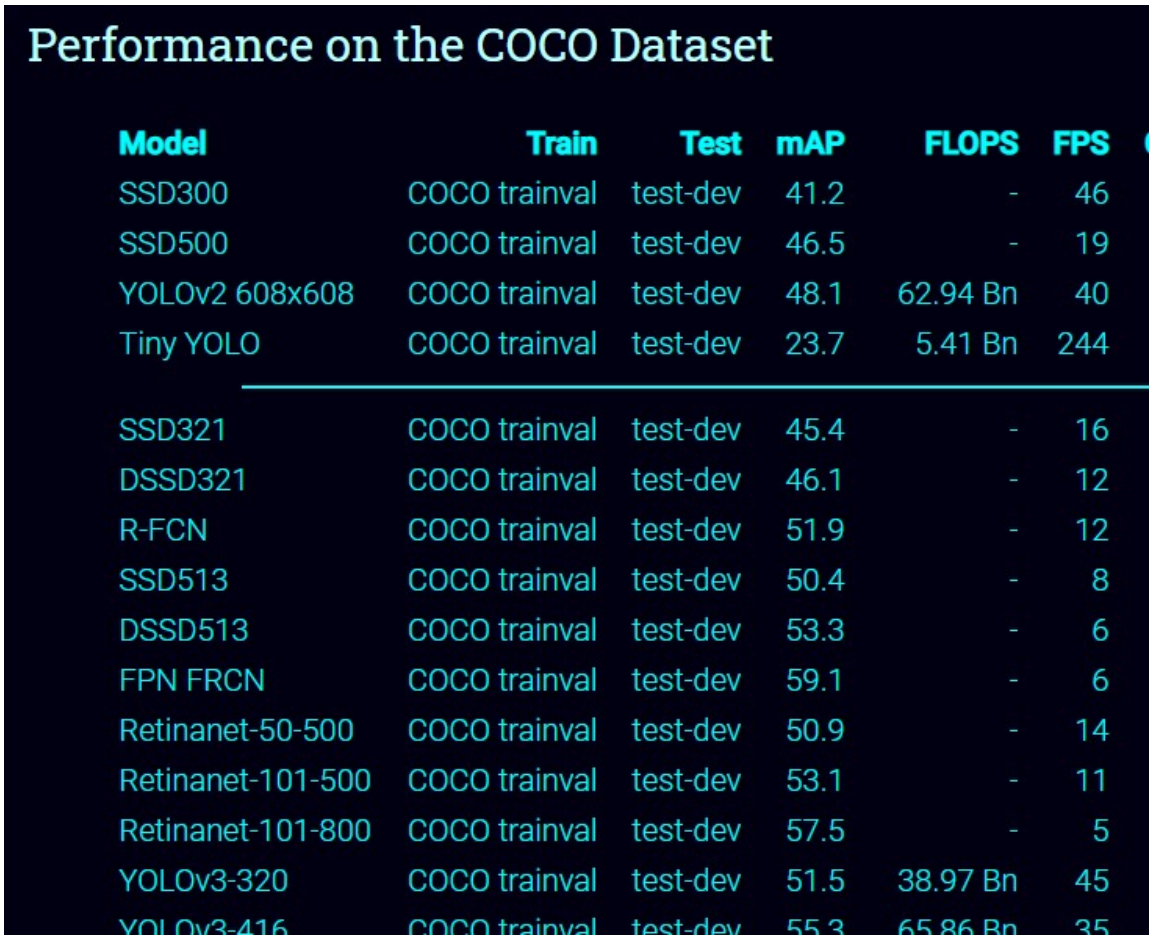
The values signify Class, x-coordinate of centre of bounding box, y-coordinate of centre of bounding box, width of bounding box, height of bounding box, respectively. Here class=20 signifies the letter ‘T’.

We opted to have only one neural net(based on YOLO) that would directly detect a sign in the image. Training a neural network from scratch is highly expensive and time-consuming, so we used the darknet53[7] architecture and the pre-trained weights. Using transfer learning, we trained this architecture to detect signs from our labeled dataset.

Why YOLO?

Other Object Detection networks such as Regional CNNs were not chosen because they take more time to detect objects as they look through an image twice. We wanted to build a model that could further be used for real-time object detection, and networks that look at the image only once seemed a better choice.

Among various other detectors that look only once, the reason for choosing YOLOv3 was based on the performance of different networks on the COCO dataset illustrated below:



Performance on the COCO Dataset

Model	Train	Test	mAP	FLOPS	FPS
SSD300	COCO trainval	test-dev	41.2	-	46
SSD500	COCO trainval	test-dev	46.5	-	19
YOLOv2 608x608	COCO trainval	test-dev	48.1	62.94 Bn	40
Tiny YOLO	COCO trainval	test-dev	23.7	5.41 Bn	244
SSD321	COCO trainval	test-dev	45.4	-	16
DSSD321	COCO trainval	test-dev	46.1	-	12
R-FCN	COCO trainval	test-dev	51.9	-	12
SSD513	COCO trainval	test-dev	50.4	-	8
DSSD513	COCO trainval	test-dev	53.3	-	6
FPN FRCN	COCO trainval	test-dev	59.1	-	6
Retinanet-50-500	COCO trainval	test-dev	50.9	-	14
Retinanet-101-500	COCO trainval	test-dev	53.1	-	11
Retinanet-101-800	COCO trainval	test-dev	57.5	-	5
YOLOv3-320	COCO trainval	test-dev	51.5	38.97 Bn	45
YOLOv3-416	COCO trainval	test-dev	55.3	65.86 Bn	35

Figure: This image was taken from www.pjreddie.com/darknet/yolo/

The mAP(mean Average Precision) for YOLOv3 Neural Network model was 60.6.

The weights learned from this training were then used to detect signs from the test images with that help of a program written in python witht the help of OpenCV.

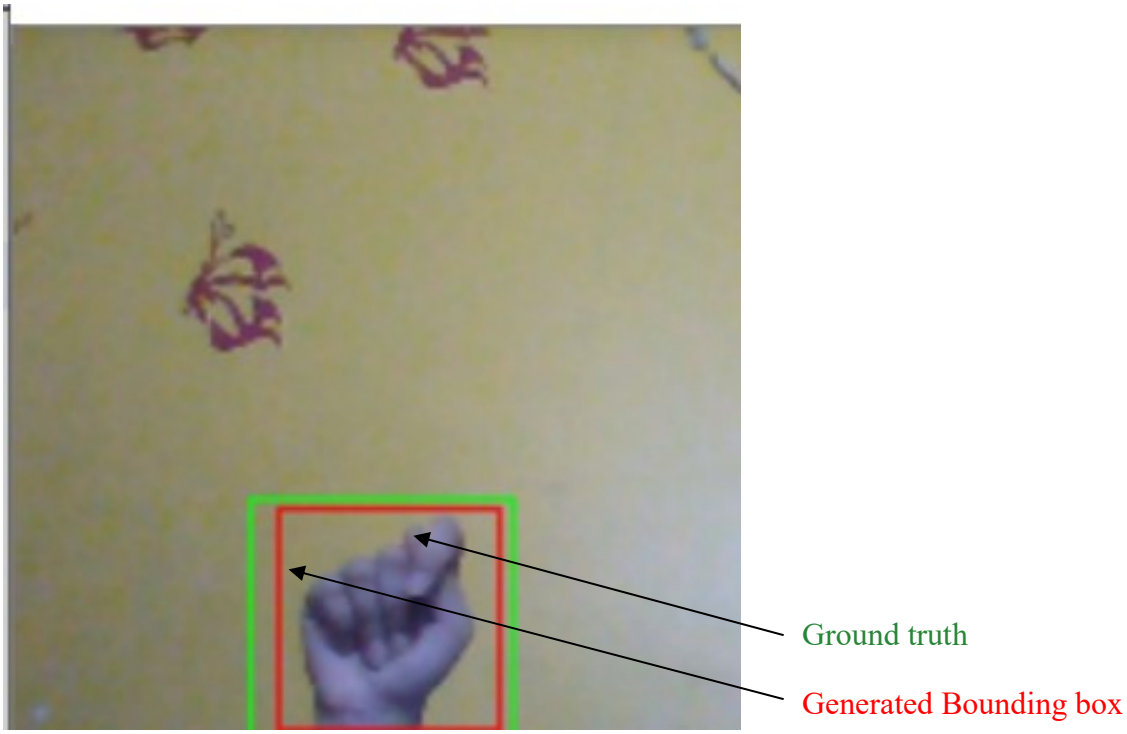
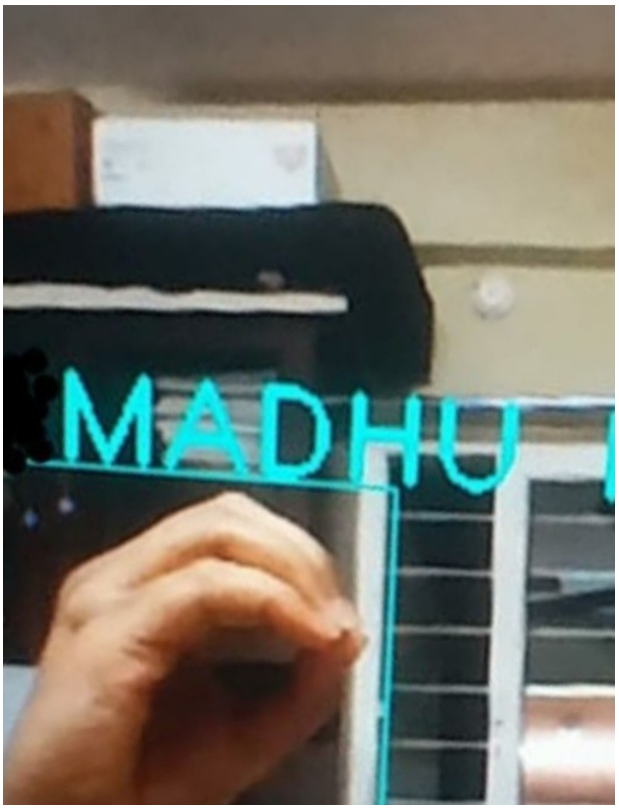


Figure 5: Accurate bounding box was generated.

RESULT

We have successfully built an image recognizing model to recognize American Sign Language (ASL) and convert it to text. Our model has achieved a mAP (mean average precision) of 60.6.



CONCLUSION

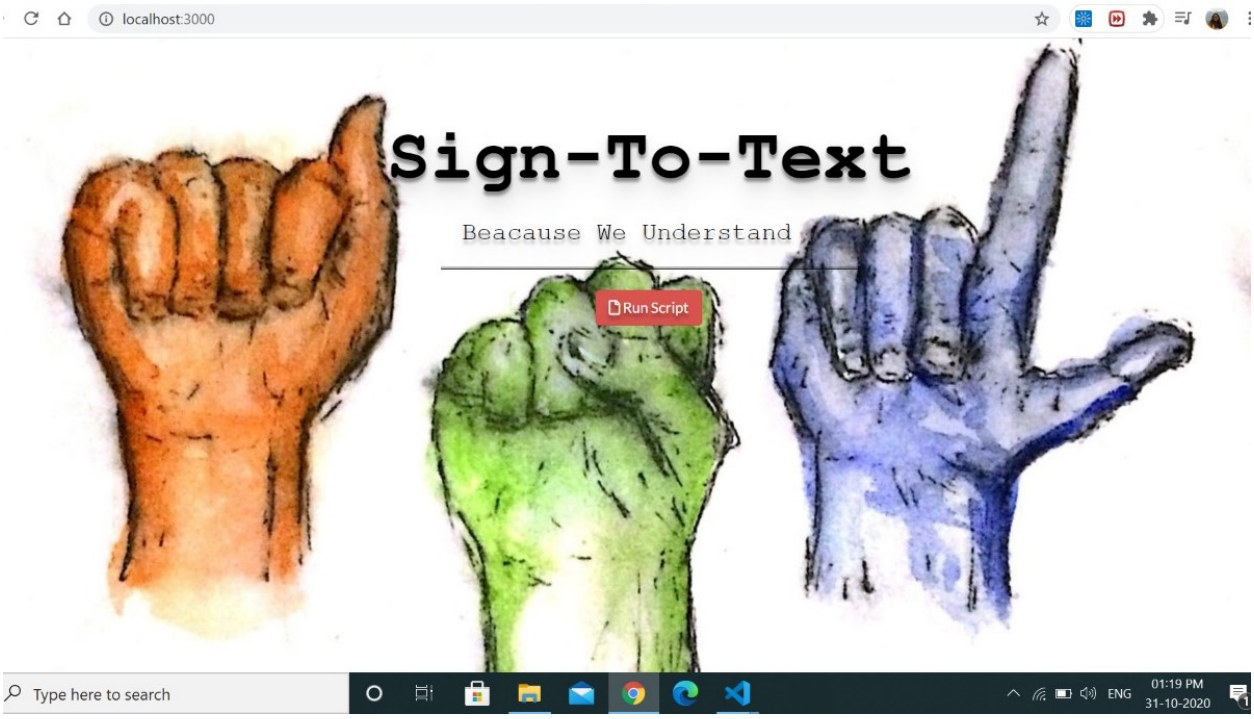
Using various concepts of image processing and object detection in images, along with the fundamental properties of image we tried to developed this system.

The You Only Look Once(YOLO) model resulted in a fast recognizer that detects signs in the American Sign Language. This recognizer can be used for real-time hand detection that can be used by people with and without hearing and speech disabilities to communicate effectively with each other.

We understand that communication is a basic human need and hence, we believe that every individual, despite their disabilities, should be able to communicate without any difficulties. Every God creature has an importance in the society, remembering this fact, let us try to include hearing impaired people in our day-to-day life and live together.

FUTURE SCOPE

We are planning to embed the recognizer into a website so that more people will be able to access the recognizer easily, thereby helping thousands of people around the world. We aim to create a website using NodeJs and JavaScript features.



REFERENCES

1. Link to colab notebook for Method2: https://colab.research.google.com/drive/1LBUIKpsWeJhXb5W4Tf4KUN_p6yeq9BhK?usp=sharing
2. Link to images, cfg files and weights: https://drive.google.com/drive/folders/11f_uEHOzS-md4pEICAaOf1AuxVJ54Os2?usp=sharing
3. Link to EgoHands website: http://vision.soic.indiana.edu/projects/egohands/
4. Link to ASL dataset in Kaggle: https://www.kaggle.com/grassknoted/asl-alphabet
5. Link to darknet Github repository: https://github.com/pjreddie/darknet
6. Link to LabelImg Github repository: https://github.com/tzutalin/labelImg
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), Las Vegas, NV, 2016, pp. 779-788, doi: 10.1109/CVPR.2016.91.