SIGN LANGUAGE DETECTION USING YOLOv8

Submitted in partial fulfillment of the requirements
for the course of CS5330

Pattern Recognition and Computer Vision

Sriram Pandi
Srinivas Peri
Rushi Shah

Under the kind guidance of
Prof. Bruce A. Maxwell
Teaching Professor

Khoury College of Computer Science
Northeastern University
April, 2023

SIGN LANGUAGE DETECTION USING YOLOv8

[*]Submitted as the final project report for CS5330

Sriram Pandi Srinivas Peri Rushi Shah

***Abstract*—Verbal communication is one of the basic and important aspects of a human life. Understanding a common language is necessary to communicate verbally similarly, people with impaired hearing and/or speech use sign language to communicate. In this paper, we present an approach for detecting and recognizing sign language alphabets using the YOLOv8 object detection model. We fine-tuned the model on a custom dataset of sign language images using the Ultra-Lytics library and trained it for 60 epochs. Our approach achieved a mean average precision (mAP) score of 0.85 on the test set, demonstrating the effectiveness of the YOLOv8 model for sign language detection and recognition. The paper also speaks about alternative approaches using Instance segmentation, Segment anything model (SAM) and corresponding initial results are presented.**

***Index Terms*—model, object detection, CNN, network, neural network**

I. Introduction

This paper focuses on implementing the object detection algorithm to detect signs in a live video feed. Object detection is done using Convoluted Neural Networks(CNN), CNN is a network with different layers which manipulate an image and identify different objects in the image and by doing this on a set of images, a CNN model can learn to recognize objects. A trained model can then be used to make predictions accurately. In this paper, the algorithm we chose is YOLOv8. YOLO is a widely used CNN model for various tasks. Over the years many versions have been released, the latest version is the v8 and it is used in this implementation. YOLO uses a very tightly developed neural network which is efficient to extract features from the images and is also not computationally expensive, meaning it does not need more resources to run the network. [1] This also helps in reducing the time taken to train the model.

II. Related work

There has been a lot of research on object detection and different applications of object detection. In particular, there has been a lot of work on detecting sign language. One of such research papers published by Premkumar, A. [2] they have collected datasets for different hand gestures and then the images from the dataset have been preprocessed for making it easier to extract and recognize features from the images. This preprocessing includes making the images greyscale, then a thresholding filter is applied and then finally a gaussian filter is applied to the images. These preprocessed images are passed to the deep learning model. The experiment mentions the use of two different models to comapre the performances of each of the model. One of the model is a simple CNN which contains two convolution layers and two maxpooling layers. This model is compared with the VGG16 model which has a lot more convoluting and maxpooling layers. The results of the training both the models was visualized, the graphs show that both the model became better as the model was trained more. The predictions improved and the accuracy was similar for both the models. As isnpiration from this research paper, we tried to implement a custom model using TensorFlow which is described in the further sections. [3]

In [4] the system utilizes a PCA analyzer model to determine the position of the hand shape. It also uses a motion chain code to understand the hand movement to make determinations about signs that require movement. They have managed to achieve an error rate of 10.91%.

[5] has utilized a number of parameters including posture of the hand, position, motion and orientation. The model uses a typical HMM with a sub-par accuracy level due to the processing happening for 51 postures, 6 orientations and 6 positions. The accuracy obtained is around 80.4%.

In [6], gestures for communication using ISL (International Sign Language) are composed of both static and dynamic hand movements, with the majority of signals involving both hands. To classify these signals, a video database containing numerous recordings is utilized. The direction histogram is used as the feature for classification due to its attractiveness for illumination and presentation invariance. Recognition is achieved using two different approaches: Euclidean distance and K-nearest neighbor measurements.

Certain models like [7] have employed a combination of Neural nets and Hough Transform for the detection of American Sign Language (ASL) using a vector feature function for comparison. The vector feature function is immune to disturbances caused by rotation and scaling, making the system highly adaptable and resilient. The methodology achieved an accuracy of approximately 92%, indicating that it is a widely accepted solution for ASL detection in the industry.

III. Dataset

Data acquisition is one of the most important steps in training a deep learning model. This means the selection of the images that contain the object in different conditions to train the model. The dataset should also be from a reliable source to ensure the quality of the images and the way they represent the object to detect. For our implementation, we decided to generate our own dataset so that we could ensure the quality of the images and also generate images while making the sign in different orientations, different lighting conditions, e.t.c.

A. Data Collection

To collect the data, a custom Python script was run which used the system camera to capture the images every five seconds. Thirty images for each pose were captured for eighteen poses, making the dataset of 506 images in total. Each captured image is shown on the screen for two seconds to make sure the object(sign) is visible in the image. The script saves the images with a unique identifier for each class(pose) followed by the number of images for that class. The program saves the images from different classes in their respective folder.

B. Labeling

Generating the ground truth for training the model is important as the model uses the ground truth to learn the object of interest in the image. For this collected dataset, the ground truth had to be made which included the size and coordinates of the bounding boxes which show the hand gesture(sign) which is the object of interest in this case. These regions in the images are then labeled with the gesture that is shown in the images figs: 1,2,3,4.

Initially the LabelImg package was used for this purpose but later to get the annotations of each image only in text format instead of xml fomrat we used the online tool 'Roboflow'to label the images. This tool not just assisted in drawing bounding boxes around the areas of interest in the images but also helped in segregating the data set into train, test and validation sets.

The images were collected for the following gestures: Correct, Drink, Eat, Goodbye, Hello, Help, House, Iloveyou, LoveMore, No, Pain, Play, Please, Smart, Stop, Thanks and Yes. The images below show some of the labeled images used as the ground truth to train the model.

C. Data Augmentation

Data augmentation is a technique used to increase the size of a dataset by applying transformations or filters to the existing images. Since we had only 506 images to increase the size of the dataset several techniques have been used to augment the collected dataset. These techniques include Horizontal Flip, Vertical Flip, Clockwise 90deg rotation, Counterclockwise 90deg rotation, 180deg rotation, and Shear angle variation. Some other techniques like adding noise, increasing the brightness of the selected regions etc. were also tried but the results were not varying much so those technique were removed. By performing data augmentation to the initial dataset of 540 images, the dataset was enhanced to 1,400 images, with 900 images used for training, 250 images for validation, and 250 images for testing. This increase in dataset size can help improve the performance of the model by providing more varied and diverse data for training.

IV. Method

To perform Real time sign language detection , the following process was followed. The first step was to collect a dataset from a reliable source, for this project we decided to generate our own dataset as described in section II. It is then important to decide the model to use for detecting the objects. A Neural Network can be developed to run the images but to improve the accuracy of this network would take a lot of training to get the most accurate weights. For that reason, we chose a pre-trained network YOLOv8 with imagenet weights. [8] Using these weights, we performed transfer learning on the model to get the most accuracy.

A. Algorithm

To recognize the signs from the collected dataset, we initially started using TensorFlow to develop a network to extract features which would be used to learn to determine the objects of interest from the images. This model would be trained on the set of annotated dataset that we created and then tested on the live camera feed. The created model took significant amount of time to train on the local machine because of the GPU accessibility issues. Therefore we moved to YOLOv8 approach.

YOLOv8 algorithm is very efficient and utilizes less resources to train. Since YOLOv8 does not have a published

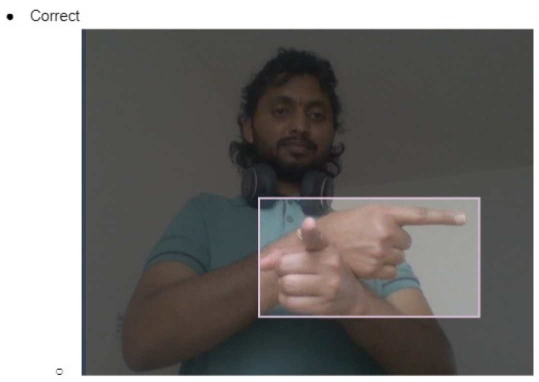


Fig. 1. Sign for correct

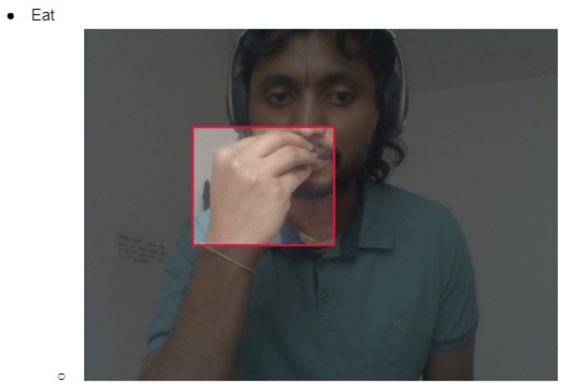


Fig. 2. Sign for eat

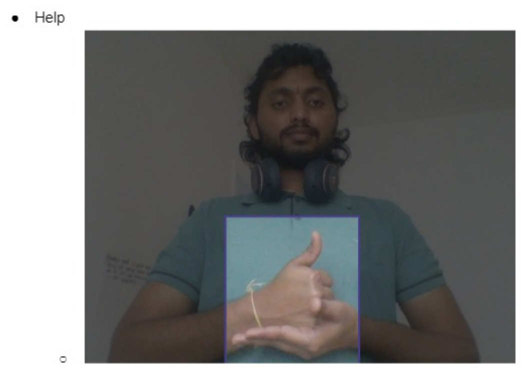


Fig. 3. Sign for help

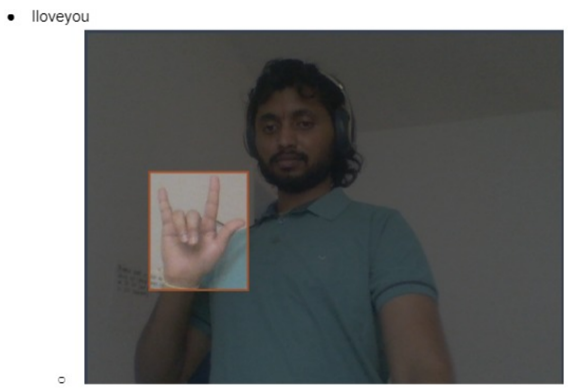


Fig. 4. Sign for I love you

paper yet, the following sections describe the network architecture used by YOLO and the changes made in the version8 as found from other resources. [9]

A usual deep learning model is made of a Convoluted Neural Network(CNN). A regular CNN consists of layers, these layers can be combination of convolution, maxpooling, dense layers, softmax and so on. The information from the image is passed through 'neurons' in the network. every neuron is assigned value according to the pixels in the image. Different layer performs different tasks and thus affect the values of these neurons accordingly. The factor by which these values are changed, are called the weights of the layer. When the model is trained,, it makes predictions, these predictions are then compared with the ground truth provided in the dataset. Comparing the predicted result with the ground truth, the model learns and changes the weights of the layers to get a more accurate prediction. This process is continued, and the model keeps on changing the weights to learn the patterns better.

1) Architecture: The CNN used by YOLO can be divided into two main parts, Backbone and the Head.

The backbone of YOLOv8 is a convolutional neural network (CNN) called DarkNet-53.It plays a crucial role in the YOLOv8 architecture by extracting high-level features from the input image that are essential for accurate object detection The DarkNet-53 network is used to extract features from the input image that are then used for object detection. It consists of 53 convolutional layers with residual connections, similar to the ResNet architecture. The residual connections allow the network to avoid the vanishing gradient problem and improve the training stability. During training, the backbone network is used to extract features from the input image, which are then passed to the Feature Pyramid Network (FPN) to generate a pyramid of feature maps with different resolutions. The FPN allows YOLOv8 to detect objects at multiple scales and resolutions.

The head of YOLOv8 consists of multiple convolutional layers followed by a series of fully connected layers.These layers are responsible for predicting the bounding boxes, objectness scores, and class probabilities for the objects detected in an image. The head of YOLOv8 consists of three main branches, each with a set of convolutional layers and output layers:

1. Objectness branch: This branch predicts the objectness score for each anchor box at each spatial location in the feature maps. The objectness score indicates the likelihood that an object is present in the anchor box, and is computed as a sigmoid activation function applied to the output of a convolutional layer.

2.Classification branch: This branch predicts the class probabilities for each anchor box at each spatial location in the feature maps. The class probabilities represent the likelihood that the object in the anchor box belongs to each of the predefined classes, and are computed as a softmax activation function applied to the output of a convolutional layer.

3.Regression branch: This branch predicts the bounding box coordinates for each anchor box at each spatial location in the feature maps. The bounding box coordinates represent the location and size of the object in the anchor box, and are computed as a linear function of the output of a convolutional layer.

The image below fig:5 shows the architecture of the YOLOv8 model. [9] [10]

B. Implementation

We used the Ultra-Lytics library to fine-tune the YOLOv8 model. Ultra-Lytics is a Python library that provides an easy-to-use interface for fine-tuning object detection models. The network of the model is shown in the image fig: 6. We trained the model for 60 epochs, using a learning rate of 0.001. We used the Adam optimizer and set the weight decay to 0.0005. To evaluate the performance of our model, we used the mean average precision (mAP) score, which is a commonly used metric in object detection. The mAP score measures the average precision across all classes at different levels of recall. The results from the trained model are shown below.

This trained model was then used to predict the gestures shown in a live camera feed. The model performed quite well in creating bounding boxes around the hand gestures and annotating the gestures with a decent accuracy. The results are described in the section below.

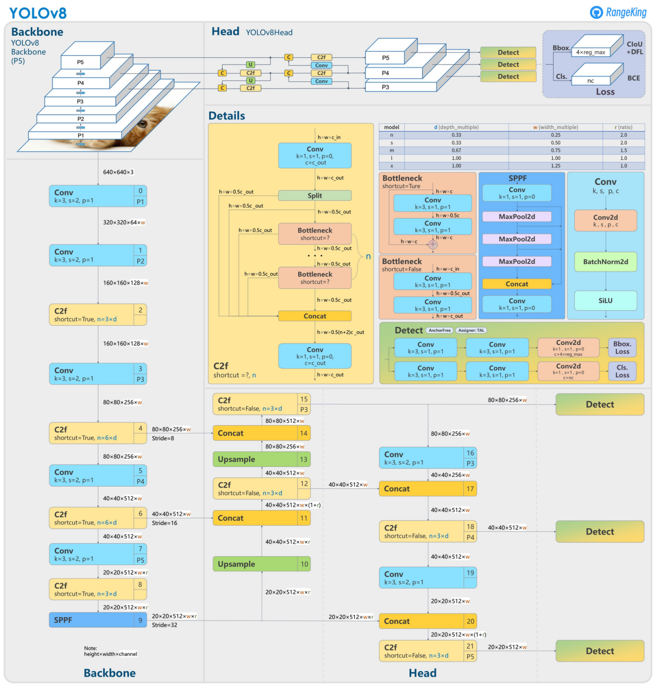
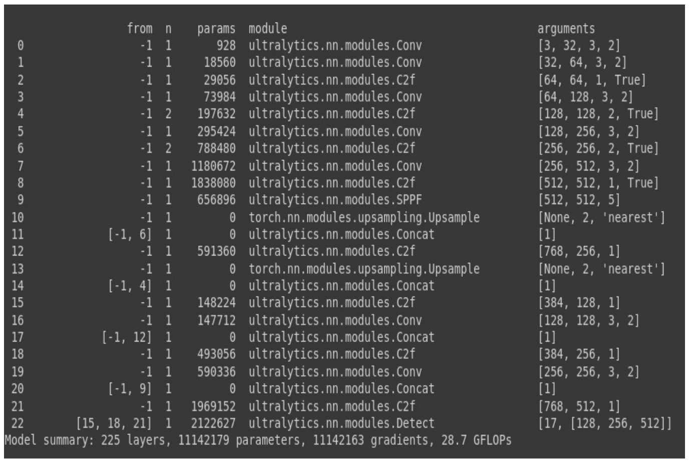


Fig. 5. YOLOv8 architecture



```
                  from  n    params  module                                       arguments
  0                  -1  1       928  ultralytics.nn.modules.Conv                  [3, 32, 3, 2]
  1                  -1  1     18560  ultralytics.nn.modules.Conv                  [32, 64, 3, 2]
  2                  -1  1     29056  ultralytics.nn.modules.C2f                   [64, 64, 1, True]
  3                  -1  1     73984  ultralytics.nn.modules.Conv                  [64, 128, 3, 2]
  4                  -1  2    197632  ultralytics.nn.modules.C2f                   [128, 128, 2, True]
  5                  -1  1    295424  ultralytics.nn.modules.Conv                  [128, 256, 3, 2]
  6                  -1  2    788480  ultralytics.nn.modules.C2f                   [256, 256, 2, True]
  7                  -1  1   1180672  ultralytics.nn.modules.Conv                  [256, 512, 3, 2]
  8                  -1  1   1838080  ultralytics.nn.modules.C2f                   [512, 512, 1, True]
  9                  -1  1    656896  ultralytics.nn.modules.SPPF                  [512, 512, 5]
 10                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 11             [-1, 6]  1         0  ultralytics.nn.modules.Concat                [1]
 12                  -1  1    591360  ultralytics.nn.modules.C2f                   [768, 256, 1]
 13                  -1  1         0  torch.nn.modules.upsampling.Upsample         [None, 2, 'nearest']
 14             [-1, 4]  1         0  ultralytics.nn.modules.Concat                [1]
 15                  -1  1    148224  ultralytics.nn.modules.C2f                   [384, 128, 1]
 16                  -1  1    147712  ultralytics.nn.modules.Conv                  [128, 128, 3, 2]
 17            [-1, 12]  1         0  ultralytics.nn.modules.Concat                [1]
 18                  -1  1    493056  ultralytics.nn.modules.C2f                   [384, 256, 1]
 19                  -1  1    590336  ultralytics.nn.modules.Conv                  [256, 256, 3, 2]
 20             [-1, 9]  1         0  ultralytics.nn.modules.Concat                [1]
 21                  -1  1   1969152  ultralytics.nn.modules.C2f                   [768, 512, 1]
 22        [15, 18, 21]  1   2122627  ultralytics.nn.modules.Detect                [17, [128, 256, 512]]
Model summary: 225 layers, 11142179 parameters, 11142163 gradients, 28.7 GFLOPs
```

Fig. 6. Model Summary

V. Experiments and Results

To experiment, we tried to make a network using TensorFlow. The network was similar to the VGG16 model, but as mentioned earlier due to the limited computational resources, the model took a very long time to train and caused the system to crash after a few epochs. While using YOLOv8 we did not face any problems and it is also not computationaly very expensive model. We were able to train the entire dataset which produced good results.

The model was imported and tested on a live video feed, the images below show the snapshots from the video feed. As the snapshots show the model was able to predict the gestures accurately. The confidence in the units of percentage is also

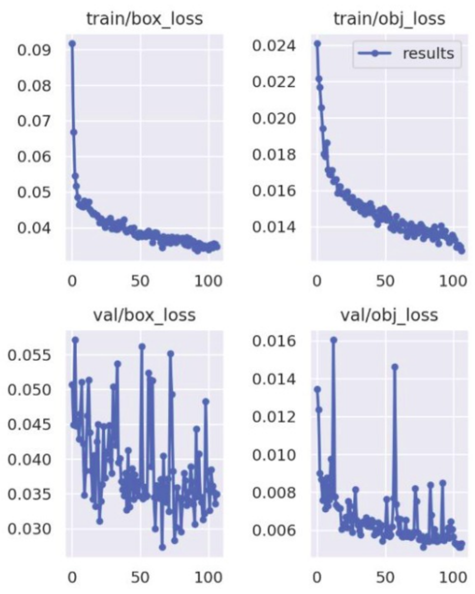


Fig. 7. Losses from the model

shown next to the prediction made by the model. The model predicts the gestures with good accuracy when the the hand is not moving, the accuracy drops when the hand is in motion. The model also tries to predict when the hand is not making any signs, these predictions would be less accurate but it still makes the boxes around the hand and makes predictions.The entire video is uploaded and the link is included in the ReadMe file. The results and the possible upgrades are discussed in the next section.

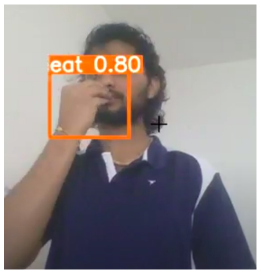


Fig. 8. result1

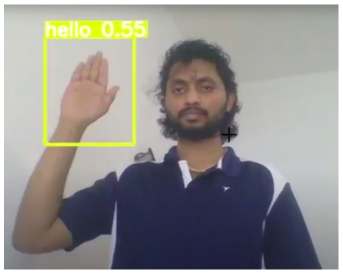


Fig. 9. result2

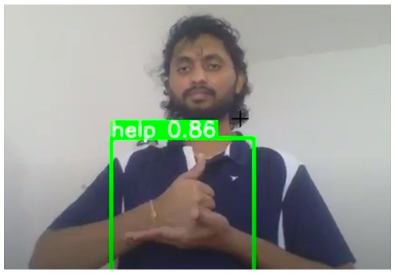


Fig. 10. result3

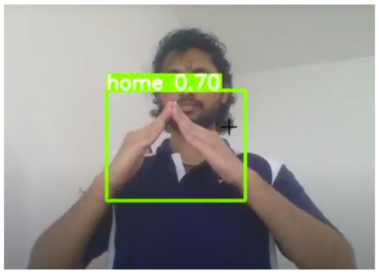


Fig. 11. result4

VI. Discussion and Summary

The results from the experiment meet our expectations. As expected from the model, the hand gestures that the model was trained for are being recognized with decent accuracy. Although there are instances where the model interprets other parts of the arm to be a hand gesture and build bounding boxes around them, giving false positives. We believe this can be solved by training the model more and with a bigger dataset. Other observations made from the results is that in the live video, when the hand was in motion and no signs were made, the model still tried to interpret something. This could also lead in false positives. We aren't sure how to solve this problem, as the better trained model would generate more false positives in this case.

The one important observation made was that the sign language contains more motion postures than static signs/symbols making it difficult to identify specific actions and accurately determine specific signs with this technique of training models with images, hence we have seen 3 possible approaches.

1) Using Action recognition where the end goal is to produce a real-time sign language detection flow by extracting key points from the hands, body, and face using MediaPipe Holistic and building an LSTM model using TensorFlow and Keras to predict the action. data on key points will be collected, trained on a deep neural network using LSTM layers, and use OpenCV to predict in real-time using a webcam.

2) Using Instance segmentation: This is a very new approach and here we would use instance segmentation to detect and recognize sign language images. Instance segmentation is a computer vision task that involves identifying and separating individual objects and their outlines in images. To achieve this, we use the Mask R-CNN model, which is a state-of-the-art model for instance segmentation. We use Roboflow to label them with polygons around the hand and fingers and then train the instance segmentation model. Here is an initial output (Fig-11) we were able to obtain, where we trained a model with smaller data set of 30 images to recognise the sign as Handsigns.

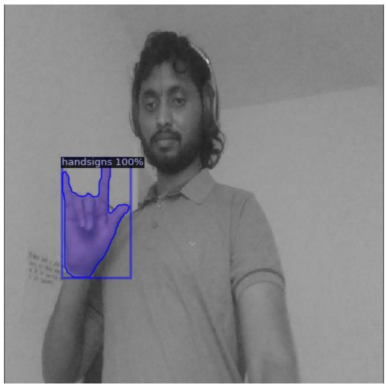


Fig. 12. Instance Segmentation

To further advance this system, there are other applications this could be used in. Like the recognition works to generate closed captions from an audio. Similarly, to advance sign language recognition, a system could be developed to interpret the gestures and make them into a sentence. This would require help of other systems too, like the sentences should make sense, they should be grammatically correct, and so on.

These are the observations from the experiment conducted and the future scopes of the project. All the code used to develop this system is included in the submitted folder. The links to the cited resources and all the other referred material is included in the references section.

References

[1] J. Terven and D. Cordova-Esparza, "A comprehensive review of yolo: From yolov1 to yolov8 and beyond," 2023.

[2] A. Premkumar, R. Hridya Krishna, N. Chanalya, C. Meghadev, U. A. Varma, T. Anjali, and S. Siji Rani, "Sign language recognition: A comparative analysis of deep learning models," in *Inventive Computation and Information Technologies* (S. Smys, V. E. Balas, and R. Palanisamy, eds.), (Singapore), pp. 1–13, Springer Nature Singapore, 2022.

[3] N. Kasukurthi, B. Rokad, S. Bidani, and D. A. Dennisan, "American sign language alphabet recognition using deep learning," 2019.

[4] T. Starner and M. Group, "Visual recognition of american sign language using hidden markov models," 05 1995.

[5] R.-H. Liang and M. Ouhyoung, "A real-time continuous gesture recognition system for sign language," pp. 558 – 567, 05 1998.

[6] A. Nandy, J. S. Prasad, S. Mondal, P. Chakraborty, and G. C. Nandi, "Recognition of isolated indian sign language gesture in real time," in *International Conference on Recent Trends in Business Administration and Information Processing*, 2010.

[7] N. Kasukurthi, B. Rokad, S. Bidani, and D. Dennisan, "American sign language alphabet recognition using deep learning," 05 2019.

[8] S. Saxena, "Using yolo to dectect sign language,"

[9] J. Solawetz, "What is yolov8? the ultimate guide.,"

[10] RangeKing