NEURAL NETWORK

PROJECT ON

“SIGN LANGUAGE RECOGNITION USING CNN, ANN AND YOLOV5”

SUBMITTED BY:

NAME: SRISTI MITRA
ROLL: 2K19/CO/389

NAME: ZISHNENDU SARKER
ROLL: 2K19/CO/450

SUBMITTED TO:
MS. REEA SACHDEVA
DELHI TECHNOLOGICAL UNIVERSITY
DEPARTMENT OF COMPUTER ENGINEERING

Shahbad Daulatpur, Main Bawana Road, Delhi – 110042

APRIL, 2022

Contents:

Candidates' Declaration

We hereby certify that the work, which is being presented in the Report, entitled **Sign Language Detection**, in partial fulfillment of the requirement for the award of the Degree of **Bachelor of Technology in Computer Science and Engineering** and submitted to the Delhi Technological University is an authentic record of out work carried out during the period from ***January, 2022*** to ***April, 2022*** under the guidance of Ms. REEA SACHDEVA.

DATE: 29/04/2022

Acknowledgement

The completion of this project till this far could not be possible without the hard work, enthusiasm, and active participation of group members contributing to this project. We are grateful to our respected project guide Ms.REEA SACHDEVA, whose insightful leadership and guidance benefited us to come this far without any hindrance. Thank you for all your support. We hope to grow better as a team and make our project as close to perfection as we can.

Furthermore, it would not have been possible without the constant research and participation of every member of this group.

Thereby,

Sristi Mitra

Zishnendu Sarker

Abstract:

Sign Language is an important part of the life of those who have speaking and hearing disabilities. Recognition of American Sign Language is very challenging using Computer Vision as Sign Language is very complex and has high intraclass variations. In this paper, ANN, CNN, YOLOV5 are used to recognize ASL Alphabets. These algorithm are useful for recognition as a deep network is expected for the ASL Alphabet Classification task. Pre-Processing steps of the MNIST and ASLAI dataset are done in the first phase. After the first phase, Various important features of the pre-processed hand gesture image are computed. In the final phase, depending on the properties computed or calculated in the initial phases, the Accuracy and AUC Score of the network model with which it can recognize the sign language Alphabets were found. We will face the same working method for the ANN, CNN model to implement it on the sign recognition system . And lastly, we will build another custom made sign recognition detection model using YOLOV5. On the supplied dataset, we were able to build a deep learning model that produced an accuracy of more than 92% for YoloV5, for the proposed CNN network was able to achieve an AUC Score of 0.9981 and an accuracy of 0.9963.

Introduction:

Approximately 18.5 million people around the world are deaf or have a voice or language impairment. They find it difficult to communicate with those around them. We can only think how much they could have communicated if they had had an easy communication medium. Speech-impaired people typically utilize ASL (American Sign Language) to communicate and make gestures with their hands. But what if they have to coordinate with someone who isn't proficient in ASL or isn't familiar with it? With this in mind, we trained our own model that has live detection of hand gestures and converts them to English language in a much faster and accurate way as possible.

- ASL: The American Sign Language (ASL) is a collection of 26 gesture signs that can be used to communicate various words from the English Dictionary. People use 19 various hand shapes to communicate in ASL, resulting in 26 American Sign Language. Because there are fewer hand forms accessible, changing the orientation of a hand shape can cause it to represent different alphabets. Letters like 'K' and 'P' are good examples. Numbers from '0' to '9' can also be expressed with hand gestures. Though no hand signals exist for specific nouns or concepts, there are a variety of facial and hand gesture indications that can be used to communicate numerous English words.

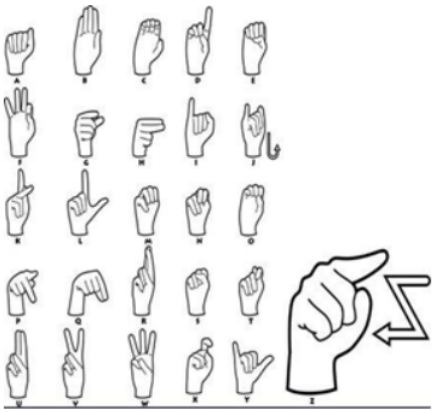
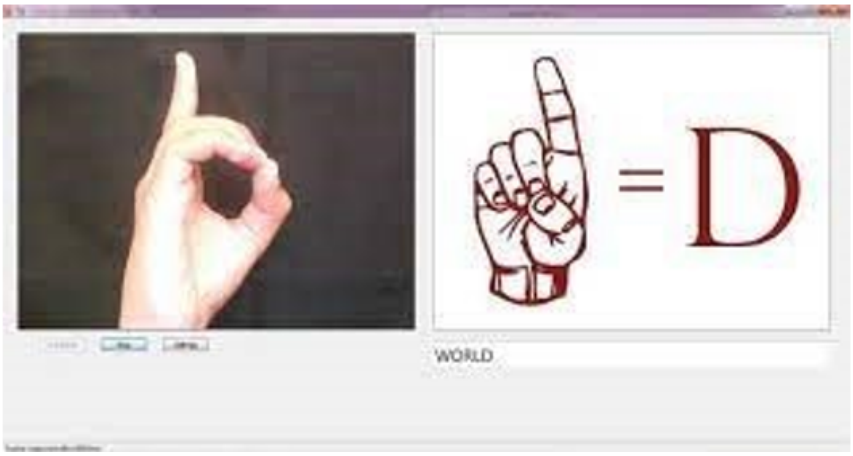
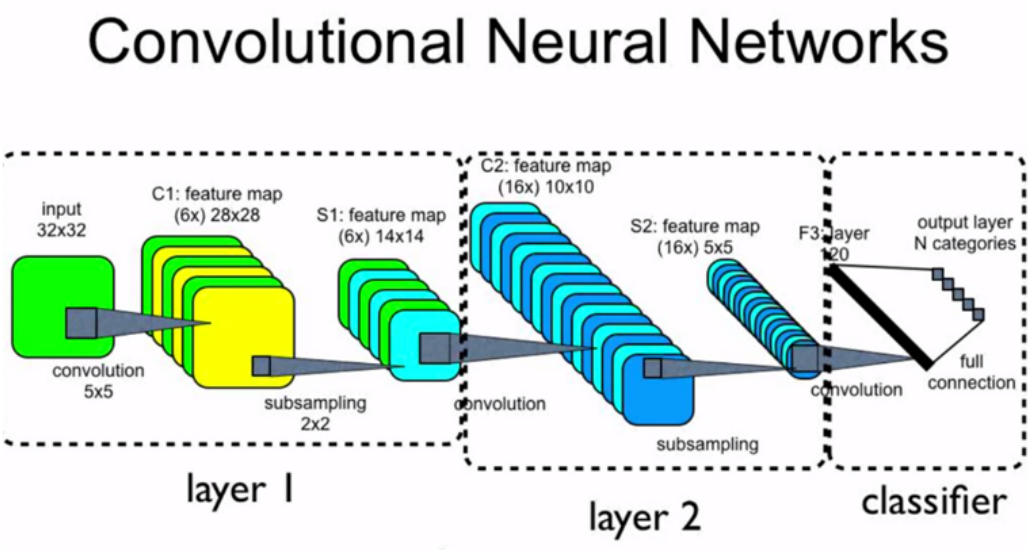


Fig. 1. Poses of Hands for each letter of the English Alphabet.

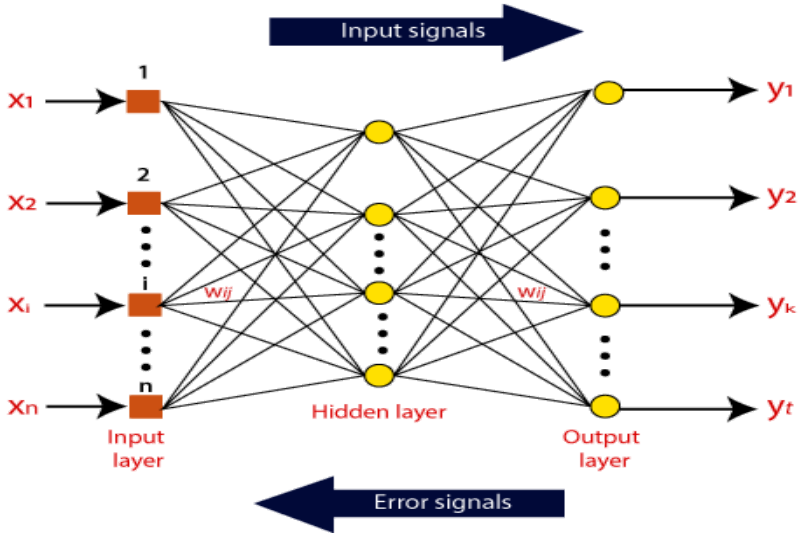
- Deep Learning: Speaking of the sign language detector, we will be using a part of Deep Learning; convolutional neural networks, that can detect the change in movements and the gestures. We will be training our model using a data set and tune it for a minimum of 85% accuracy. Hyper tuning of the model will be done according to our needs.



- CNN: A convolutional neural network (CNN) is a category of deep neural networks, most typically applied to research visual imagination. When we hear the word 'neural network' we expect it to be about matrix multiplications however that's not the case with CNN. It uses a special technique known as Convolution. In arithmetic convolution could be a computation on 2 that produces a 3rd function that expresses however the form of 1 is changed by the opposite. Bottom line is that the role of the CNN is to reduce the images into a form that is easier to process, without losing features that are critical for getting a good prediction.

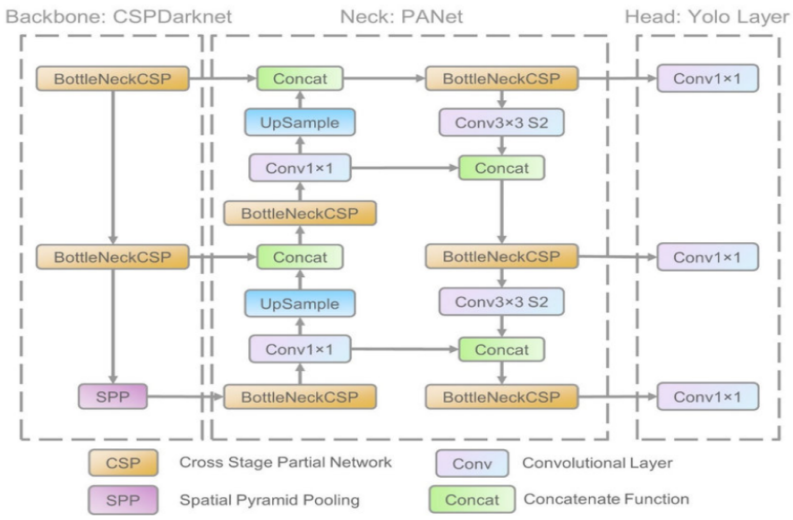


- ANN: Basic and advanced ideas of ANNs are provided via the Artificial Neural Network Tutorial. Artificial neural networks are a branch of artificial intelligence inspired by biology and fashioned after the brain. A computational network based on biological neural networks, which create the structure of the human brain, is typically referred to as an artificial neural network. Artificial neural networks also feature neurons that are linked to each other in different layers of the networks, just as neurons in a real brain. Nodes are the name for these neurons.



- YoloV5: You Only Look Once is referred to as YOLO in an abbreviation. Version 5, which Ultralytics released in June 2020, is now the most sophisticated identification algorithm on the market. It is a cutting-edge convolutional neural network (CNN) that accurately recognizes objects in real time. This method processes the entire image using a

single neural network, then divides it into pieces and forecasts bounding boxes and probabilities for each component. The predicted probability weighs these bounding boxes. The technique "only looks once" at the picture since it only does one forward propagation loop through the neural network before making predictions.



Motivation:

Many social issues in our culture have been kept quiet simply because people are not yet ready to openly discuss them and accept them as normal. We chose this project as a method to not only improve our project development skills, but also to take a little step toward addressing a subject that is still taboo in many societies. Our website was created in response to the following question: "Why not make it easier and more efficient for people with speech issues to voice their opinions?"

This project was inspired by the stories of numerous people who have been marginalized in various sectors due to a flaw that was not their fault. There are no fields in which a person with a speech handicap cannot succeed. Just because they don't speak doesn't mean they don't have valuable insights that can improve the world. Let's not forget the 20 million people who have contributed to make the world a better place.

Previous work on the project:

Many ways have been put forward to accomplish the problem of recognizing sign language gestures. Some prior work involves the use of SVM. SVM was earlier used to classify gestures in SASL] and parallel hidden Markov models were used as well.

Convolutional neural networks from depth maps are used to recognise fingerspelling in sign language in real time. This piece focuses on static American Sign Language fingerspelling. a technique for putting into practise a sign language to text/speech conversion system without the need of portable gloves and sensors, which involves continually recording the gestures and turning them into voice. Only a few photos were taken for identification using this approach. the layout of a device to let physically unable people communicate.

Creation of a physical disability communication assistance. The MATLAB environment was used to construct the system. The training phase and the testing phase make up the majority of its stages. The author employed feed-forward neural networks during the training stage. The issue here is that MATLAB is not very effective, and it is also challenging to integrate the concurrent qualities as a whole.

For the Deaf and Dumb, American Sign Language Interpreter System Twenty of the twenty-four static ASL alphabets could be recognised using the approaches presented. Due to the occlusion issue, the letters A, M, N, and S could not be discerned. Only a few photos have been utilised.

An approach to classify Sign language is proposed. The model proposed recognizes ASL alphabets gestures with a success rate of 86.67%. A real-time interaction system between humans and computers based on hand gestures is developed. In this model, images were pre-processed by applying various operations like hand color filtering, morphological transformations, etc. Later CNN was used on the dataset along with YoloV5.

A Kalman estimator was used so that whenever a gesture is identified the mouse pointer will move in response to it. Lastly, the system was able to achieve 99.8 % of accuracy on 16 gestures.Singha made a dataset that contains 240 images of 24 signs present in Indian Sign Language or ISL.

Then the Eigenvalues were extracted from the images in the dataset and were classified based on their Euclidean Distance. This system was able to achieve 97% accuracy.

Project Description:

The machine learning model that we have used three models (ANN, CNN, YOLOV5). They are the algorithms to implement deep learning in any dataset. The dataset has been made by capturing images, annotating them, by labeling them with proper classes using proper pixels of matrices. Here, we implement this project by using two different dataset, where Sign Language MNIST used for ANN, CNN and Americal Sign Language Letter image dataset for YOLOV5.

The dataset has been then turned into an image that the machine learning model can use by proper feature selection and boundary detection in all the cases. The machine learning model learns from the patterns of the dataset and then predicts the image that has been sent as an input from the output of the previous training and test datasets.

- **1. Creating Dataset :** In this project we use the photos of American Sign Language (ASL). and For yolov5, We build the dataset with ASL and level them with yolov5 format.

- **2. Deep learning Model :** Convolutional neural networks, a type of artificial neural network used to analyze visual information, will be the foundation for the deep learning model. The model will be trained on a set of data and is expected to achieve an accuracy of at least **85%.** The Python programming language will be used to create this model.

- **3. Visualization of the project :** The model we trained with CNN will be checked as a computer vision where we use a webcam to take the videos and after visualizations the model will provide the information about the sign detection.

- **4. Model Data Flow :**
    1. Images used for training model
    2. Using image processing to better visualize the feature if the image
    3. Train the model and download the weight
    4. Applying gaussian blur to the input image for better visualization
    5. Using the weight for predicting the user's sign language by the trained model

Project Requirments:

# Base:-	# Logging :-	# Plotting :-	# Export :-	# Extras :-
matplotlib>=3.2.2 numpy>=1.18.5 opencv-python>=4.1.2 Pillow>=7.1.2 PyYAML>=5.3.1 requests>=2.23.0 scipy>=1.4.1 torch>=1.7.0 torchvision>=0.8.1 tqdm>=4.41.0	tensorboard>=2.4.1 # wandb	pandas>=1.1.4 seaborn>=0.11.0	coremltools>=4.1 # CoreML export # onnx>=1.9.0 # ONNX export # onnx-simplifier>=0.3.6 # ONNX simplifier #scikit-learn==0.19.2 #CoreML quantization # tensorflow>=2.4.1 # TFLite export #tensorflowjs>=3.9.0 # TF.js export # openvino-dev # OpenVINO export	albumentations>=1.0.3 # Cython # res: https://github.com/cocodataset/cocoapi/issues/172 # pycocotools>=2.0 # COCO mAP # roboflow # FLOPs computation

Tools and Technologies used:

- Python
- Miniconda
- Maxpooling
- Keras
- Google Collab
- Jupyter Colab
- CUDA

Methodology :

Data collection is one of the most important aspects of any exploration. It is critical to collect data that is both relevant and meets the research's objectives. Readings that duplicate all scenarios, including severe cases, should be included in the data. This allows for more in-depth and realistic observations. The data gathered is the bedrock of any study and should be handled with extreme caution. There we use two dataset to train our models, where Sign Language

MNIST dataset from kaggle which is used in ANN, CNN algorithm and Americal Sign Language letters image dataset from roboflow.

- **Loading and Preprocessing Dataset:** We started by importing the dataset to work upon and making it suitable to perform various algorithms to predict sign language alphabet labels correctly. In the given dataset, we already have separate training and tests. We visualized our dataset, and it came out to be balanced. We also store the labels of each image, which is mapped with their respective labels in a list, and generates a random image to perform class label verification. We performed a grayscale normalization to reduce the effect of illumination's differences. The Dataset used:

    1. **Sign Language MNIST:** we decided to design the dataset to match the American Sign Language classic dataset. Therefore, in the dataset, we have labeled each element as training or test cases ranging from 0 to 25, which act as a matched map to all the alphabets in the English language ranging from A-Z. The dataset contains 1704 colour images where as modification and expansion strategy 15%brightness, 5% random pixelation and 3 degree rotation used. This is approximately half the size of the ASL dataset, but it replicates it accordingly as every header row has a label of pixel to pixel 784, which represents the grayscale values of 28x28 pixel images in which the values range from 0-255. The data collected in the ASL data is varied by having the same signs made by different subjects in different backgrounds. Moreover, the images were also cropped in various ways; however, the desired hand region was left unchanged. The wide variety of training sample values are evenly distributed, as can be seen from the visualization. Moreover, the CNN converges faster on [0..1] data than on [0..255]. After applying normalization, we split the training set into parts, which is the training set and validation set. So now, we have three sets of data : training set, validation set, and test set. To feed this data into the CNN model, we converted the single-dimensional data into CNN accepted format of 3 dimensions. Thus, we accomplished this by converting each row of 784 columns into (28,28,1) matrix.

    2. **American Sign Language Letters Image:** Here, the dataset contains total 720 images where 504 images are for training, 144 for validation and 72 images for the test sets. No extra modification or strategy used for expansion of database. The dataset a training result of 90% precision. The American Sign Language Letters dataset contains bounding boxes for each ASL letter and is an object detection dataset. Data scientist David Lee, who has an emphasis on accessibility, selected and made the dataset available for usage by everyone. Here it has total 26 class for A to Z.

- **Data Augmentation:** In Data augmentation from one image, we generate more images. Therefore, we can have a variety in our dataset, and it is also used to increase the size of the dataset to avoid overfitting. Our data augmentation includes random rotation ranging 10 degrees and random height and width shift in the range of 0.1, random zooming, and random flips. This was implemented using Keras image pre-processing module.

- **Model Building:** We build the model using the combination of convolution, max pooling, and flattening layers in Keras. We also used batch normalization and dropout layers to prevent the model from overfitting the data. We used a callback to determine if our validation set Accuracy did not increase even after 2 (patience level) consecutive epochs during the model fitting stage. Then, we will alter our learning rate by a factor of 0.5.

- **Independent variables and Dependent variables:** Independent variable- number of images, pixel distribution of images which is stored in NumPy array. Dependent variables are- feature map after each convolution layer output, the volume of feature map which is input to the next layer, probabilities of labels which are used for classification.

- **Parameters and Hyper Parameters:** Parameters like kernel size, astride of filter, padding, number of features/filters, filter size, activation function, a connection between two layers, pooling type, number of labels. Hyperparameters like coefficients in the learning rate, dropout rate, batch size, number of the hidden and dense layer in architecture, number of epochs.

- **Performance Measure:** Performance measures are used to check the correctness of our model. The majors we are going to use are Accuracy on both training and validation dataset, loss in both validation and training dataset, Precision, Recall, and F1 measure of all 25 classes.

- **Validation Method:** We are using the Holdout validation technique to evaluate our model. We initially have a training and testing set. Our dataset contains 704 images. 20% of total images (144) are present in our testing set and the rest in the training se. We split our training set into two parts, one for training our model and another part which is a validation set for validating our model. Later we will analyze our model based on predictions predicted by our trained model on the test set and obtain its accuracy, precision, recall, F1 Score and AUC Score to see its performance.

Implementation of Models：

● Yolov5:

The object detection method YOLO, which stands for "You Only Look Once," separates images into a grid structure. Each grid cell is in charge of detecting items within itself. Because of its speed and precision, YOLO is one of the most well-known object recognition algorithms.

➔ **Requirements:** We will need Python >= 3.8 and PIP in order to follow this guide. The rest of the requirements are listed in './requirements.txt'

➔ **Installation:** We first have to clone the repository then enter the root directory and lastly install the required packages from your cloned repository root directory.

➔ **Packaged environment:** A quick and hassle-free setup YOLOv5 has been packaged with all dependencies* for the following environments, *including CUDA/CUDNN, Python and PyTorch. Google Colab and Kaggle notebooks with free GPU, Amazon Deep Learning AMI. Docker Image.

➔ **Inference:**

➢ From cloned repositories:

To get started with object detection using the latest YOLO models, run this command from your repository root directory. Results are saved to './runs/detect'

$ python detect.py --source OPTION

We have to replace OPTION with our selection, to detect from:

Webcam : (OPTION = 0) For live object detection from our connected webcam

Image : (OPTION = filename.jpg) Creating a copy of the image with an object detection overlay

Video : (OPTION = filename.mp4) Creating a copy of the video with an object detection overlay

Directory : (OPTION = directory_name/) Creating a copy of all file with an object detection overlay

Global File Type (OPTION = directory_name/*.jpg) Creating a copy of all file with an object detection overlay

RTSP stream : (OPTION = rtsp://170.93.143.139/rtplive/470011e600ef003a004ee33696235daa) For live object detection from a stream

RTMP stream : (OPTION = rtmp://192.168.1.105/live/test) For live object detection from a stream

HTTP stream : (OPTION = http://112.50.243.8/PLTV/88888888/224/3221225900/1.m3u8) For live object detection from a stream
The following file formats are currently supported:

Images: bmp, jpg, jpeg, png, tif, tiff, dng, webp, mpo
Videos: mov, avi, mp4, mpg, mpeg, m4v, wmv, mkv

➢ From Pytorch hub:
Inference can be run directly from PyTorch Hub without cloning the repository. The necessary files will be downloaded into your temporary directory.

Here is an example script that uses the latest YOLOv5s model and the repositories example images.

```
import torch

# Model
model = torch.hub.load('ultralytics/yolov5', 'yolov5s')

# Images
dir = 'https://github.com/ultralytics/yolov5/raw/master/data/images/'
imgs = [dir + f for f in ('zidane.jpg', 'bus.jpg')]  # batch of images

# Inference
results = model(imgs)
results.print()  # or .show(), .save()
```

- **CNN:**

Our model is sequential since we are initializing this deep learning model as a sequence of layers. The information is propagated from the input layer to the hidden layer to the output layer through the model. This network employs a mathematical operation called convolution. The primary purpose of convolution is to find features in our image using a feature detector and put them into a feature map and having them in a feature map, and it preserves the spatial relationship between pixels, which is very important for us.

The first step towards building a model for our work is to initialize the model. Since our model is sequential,we initialized it as sequential. The next step is to create our first layer of CNN to perform convolution operations. We created the input layer of our CNN. We have taken 32

feature detectors of 3x3. The layer takes input into a batch of images. In our case, the batch size is 32. The shape of our image for the input layer is (28,28,1), where the height and width of the image are 28, and there is one channel as we grayscaled our images. Each feature detector produces a feature map after convolution operation over the image with an output volume of 26x26x1. We have taken the default value of padding that is 0 and stride as 1. After the convolution operation, the ReLU activation function (Figure 6.) is applied to increase non-linearity in our CNN model. If the value is greater than 0, it passes the input; otherwise, it returns 0.

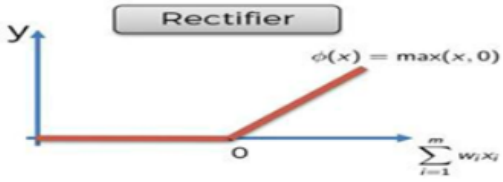


Fig: ReLU activation function

Now we have performed Batch Normalization to keep the output of the layer of the same range to get an unbiased result. After this, the output of our first convolution layer of volume 26x26x32 is passed for the max-pooling operation to the next layer. Max pooling help to get rid of unnecessary features and account for their spatial or textual or any kind of distortion. Pooling also helps in preventing any kind of overfitting as it avoids few parameters while performing a pooling operation to get a pooled matrix. Max pooling is used to focus only on the relevant information from the image by extracting only higher values from a portion of input by using an empty feature detector. Pooling is also called downsampling. We are taking stride as two and filter feature detector size as 2x2. As a result, the height and width are reduced by a factor of 2.Thus, the output volume will be 13x13x32. Now we have a second layer, which consists of 1 convolution 2d layer and one max pooling. The input of this layer is the output of the previous layer, which is 13x13x32. The kernel size is 3x3, with 64 different feature detectors. So, the output volume is 11x11x64. Now we again applied Batch Normalization and passed the output volume 11x11x64 as input for Max Pooling, and we get the output as 5x5x64. Now we have the last layer, which is the same as all previous layers consisting of 1 conv2d layer and one max pooling layer. So, the operations performed will be similar, but in the last layer, the number of feature detectors we took is 128. So, the output of the final layer will be 1x1x128[8]. Now we will perform a flattening operation to make it suitable for ANN's input. So, after flattening, it will be converted into a 128x1 vector, and it will be fed to the dense layer (Hidden layer in ANN) with some initialized weights and bias. The output of the dense layer is passed through the ReLU activation function to increase the non-linearity in the model. Now we have used a dropout layer with a dropping rate of 25%, which means 25% of random neurons are switched off so that we can prevent overfitting of our model[12]. Now the output of this hidden or dense layer is fed to

the final or output dense layer with 25 neurons. This layer has 25 neurons representing 25 classes. The activation function here we have used is the Softmax activation function. Here we could have used the sigmoid function as the sigmoid function is the heart of the probabilistic approach, but the sigmoid function is good only when we classify between 2 classes. So, when there are more classes, softmax is used, and softmax ensures that the sum of the probability of the output is equal to one, which makes it easy to understand.

probability array of all 25 classes, and the probabilities are checked with the actual output, and the loss is calculated using sparse categorical cross-entropy[7]. Here we have used sparse categorical cross-entropy because we have labelled as targets. Finally, we compile our model, and to minimize our cost function, we use 'adam' optimizer (stochastic Gradient Descent Method) to attain a global minimum. 'adam' optimizer is used with a learning rate of 0.001 and values of b1=0.9 and b2=0.999. During the backpropagation in the model, weights in the dense layer, as well as weights present in the feature detector changes using the local gradient if not useful for the model. The total trainable parameters to be calculated are 171,993, as seen in Figure , representing each layer with its output shape and trainable parameters.

Layer (type)	Output Shape	Param #
conv2d (Conv2D)	(None, 26, 26, 32)	320
batch_normalization (BatchNo	(None, 26, 26, 32)	128
max_pooling2d (MaxPooling2D)	(None, 13, 13, 32)	0
conv2d_1 (Conv2D)	(None, 11, 11, 64)	18496
dropout (Dropout)	(None, 11, 11, 64)	0
batch_normalization_1 (Batch	(None, 11, 11, 64)	256
max_pooling2d_1 (MaxPooling2	(None, 5, 5, 64)	0
conv2d_2 (Conv2D)	(None, 3, 3, 128)	73856
batch_normalization_2 (Batch	(None, 3, 3, 128)	512
max_pooling2d_2 (MaxPooling2	(None, 1, 1, 128)	0
flatten (Flatten)	(None, 128)	0
dense (Dense)	(None, 512)	66048
dropout_1 (Dropout)	(None, 512)	0
dense_1 (Dense)	(None, 25)	12825

Total params: 172,441
Trainable params: 171,993
Non-trainable params: 448

Fig: Training parameter CNN

- **ANN:**

We are using keras which is an open-source library to build our model. So, initially we imported keras and all the other dependencies. Then, we have initialized our model as sequential model. Input layer of our network expects rows of data with 784 variables. We have input dimension as 784 which is equal to the number of columns in the dataset . We have used four hidden layers each of which has 404 nodes and uses the Rectified Linear Unit (ReLU) Activation Function because of its fast computation capabilities. ‘Dense’ at each layer signifies that layers will be fully connected in our model. We have also added dropout between the hidden layers for prevention of overfitting. Along with it, we have also added batch normalization between the layers for faster working of our model. Batch Normalization adds a few new layers in our neural network which normalizes the values corresponding to the input layer.

At the output layer , we have 25 nodes and we have used Softmax activation function .We have used sparse categorical cross-entropy because we have labelled them as targets. Finally, we compile our model, and for minimization of cost function , we use the 'adam' optimizer (stochastic Gradient Descent Method) to attain a global minimum. 'Adam' optimizer is used with a learning rate of 0.001 and values of b1=0.9 and b2=0.999. Along with it, we have also used ReduceLROnPlateau to reduce the learning rate so that our model shows improvement. Then, we trained the model on a training dataset.

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
dense (Dense)                (None, 404)               317140
_________________________________________________________________
batch_normalization (BatchNo (None, 404)               1616
_________________________________________________________________
dropout (Dropout)            (None, 404)               0
_________________________________________________________________
dense_1 (Dense)              (None, 404)               163620
_________________________________________________________________
batch_normalization_1 (Batch (None, 404)               1616
_________________________________________________________________
dropout_1 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_2 (Dense)              (None, 404)               163620
_________________________________________________________________
batch_normalization_2 (Batch (None, 404)               1616
_________________________________________________________________
dropout_2 (Dropout)          (None, 404)               0
_________________________________________________________________
dense_3 (Dense)              (None, 404)               163620
_________________________________________________________________
dense_4 (Dense)              (None, 25)                10125
show more (open the raw output data in a text editor) ...
=================================================================
Total params: 822,973
Trainable params: 820,549
Non-trainable params: 2,424
```

Fig: Training parameter ANN

Result and Analysis :

- **For YOLOV5:**

  The graphs are given below come from the tensorboard from our model . As we can see that, here the model has 4 graphs metrics where they are presenting the mAP_0.5, mAP_0.5 to 0.95, precision and recall for 200 epochs. Where we can see that , as the number of epochs reaches 200 the value we get is near 1 and that means that our model gives more precise values here. And also we can see that, when we recall the datas it also fluctuates between .90 to 1 . As we can see here , these are the graphs of our training loss, box loss and object loss. We can see there that, as we are going near 200 epochs, the losses of the values and variables are decreasing and getting near 1.

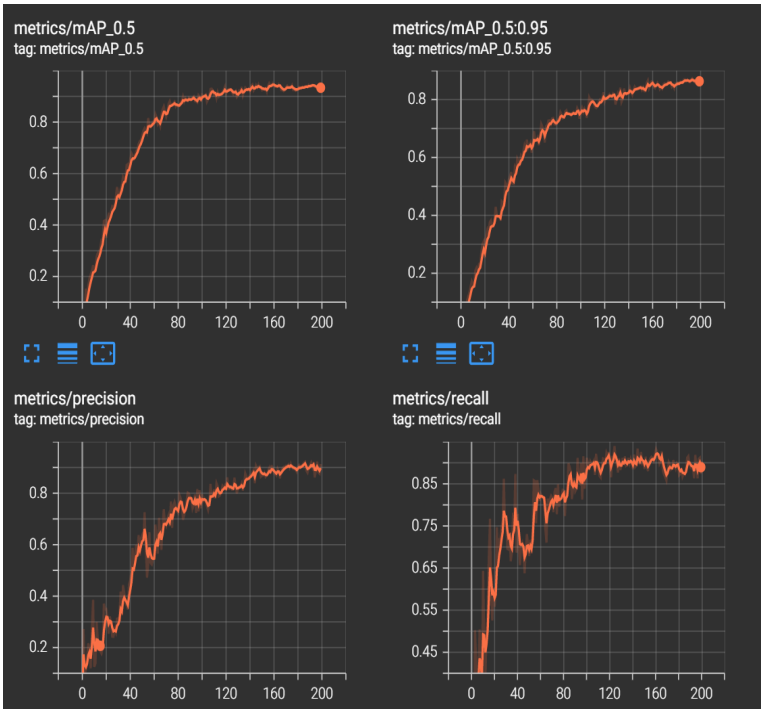


Fig : Accuracy Graph

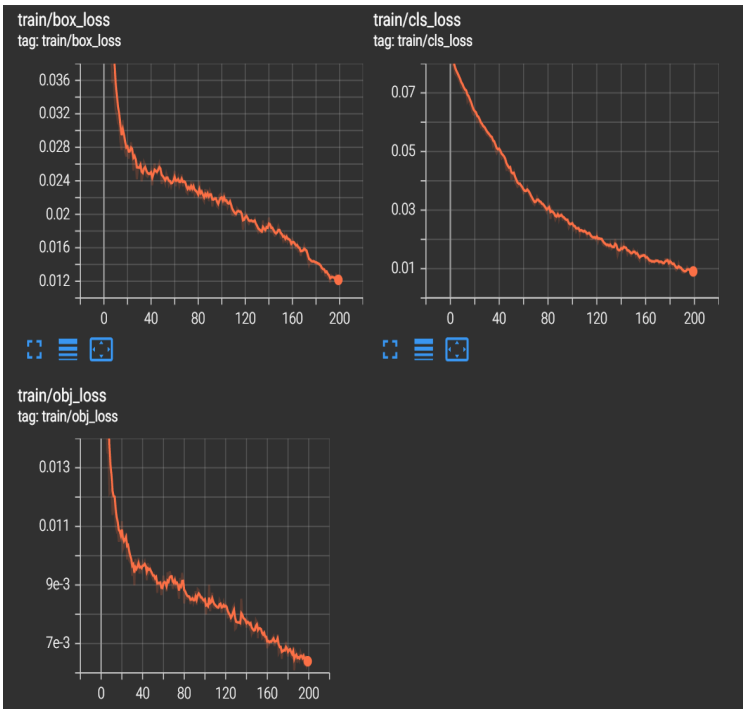


Fig : Loss Graph

Below there we can see, our model took 1.071 hours to train the model weight for 200 epochs where can see all the classes for the alphabets. We took 144 valid image set for every alphabet and levels them. Also the below graph is showing the precision values, recalling values and also the values of mAP 0.5 and mAP_0.5 to 0.95 . Were are getting almost 90 percent precision values on average and also more than 92 % accuracy in average from the model. Here we are adding some figures of the webcam visualizations and here we use real time sign language detection along with still photos through mobile device. Also our model is able to check multiple sign languages together.

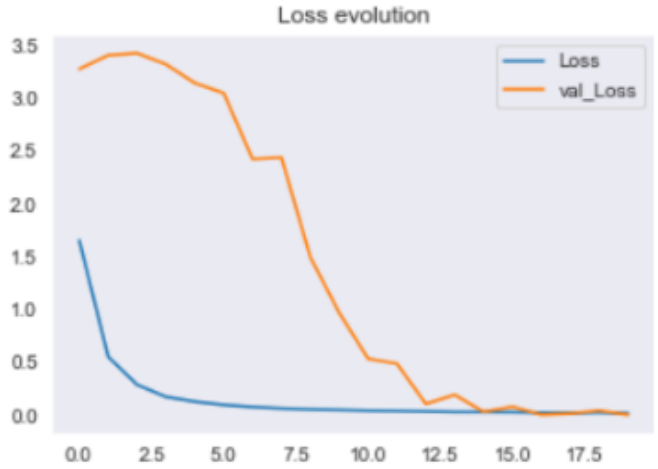
Code + Text

```
200 epochs completed in 1.071 hours.
Optimizer stripped from runs/train/exp/weights/last.pt, 14.4MB
Optimizer stripped from runs/train/exp/weights/best.pt, 14.4MB

Validating runs/train/exp/weights/best.pt...
Fusing layers...
Model summary: 213 layers, 7080247 parameters, 0 gradients, 16.0 GFLOPs
               Class     Images     Labels          P          R     mAP@.5 mAP@.5:.95: 100% 5/5 [00:03<00:00,  1.65it/s]
                 all        144        144       0.92      0.863      0.944      0.872
                   A        144          5      0.964        0.6      0.818      0.768
                   B        144          9          1      0.841      0.995      0.902
                   C        144          3      0.999          1      0.995       0.94
                   D        144          6      0.936          1      0.995      0.974
                   E        144          4      0.991          1      0.995      0.995
                   F        144          8          1      0.924      0.995      0.983
                   G        144          5      0.979          1      0.995      0.934
                   H        144          9          1      0.907      0.995      0.977
                   I        144          2      0.408        0.5      0.283      0.257
                   J        144          8      0.998          1      0.995      0.713
                   K        144          6       0.92      0.667      0.927      0.884
                   L        144          4      0.794          1      0.995      0.937
                   M        144          8          1      0.625      0.995      0.918
                   N        144          4      0.568          1      0.796      0.765
                   O        144          7      0.994          1      0.995      0.935
                   P        144          7          1      0.571      0.995      0.734
                   Q        144          4      0.834          1      0.995      0.914
                   R        144          7          1          1      0.995      0.979
                   S        144          4      0.982          1      0.995      0.995
                   T        144          6          1      0.668      0.942      0.901
                   U        144          7          1      0.908      0.995      0.889
                   V        144          5      0.718        0.6      0.881      0.812
                   W        144          3      0.998          1      0.995      0.929
                   X        144          1      0.853          1      0.995      0.796
                   Y        144          8          1       0.63      0.982      0.884
                   Z        144          4      0.989          1      0.995      0.952
Results saved to runs/train/exp
```

Fig: result of each alphabets (YOLOV5)

- **For our CNN model:**



We trained our model on 20 epochs with a batch size of 512. The maximum accuracy obtained on the training set was 99.60%, and the invalidating set was 99.95%— epochs. The accuracy graph is shown for training and validation accuracy, training and validation loss, respectively.

	Accuracy	Precision	Recall	AUC Score	F1 Score
Result:	0. 9963	0.99	1.00	0.9881	1.00

Table: Accuracy, Precision, Recall, AUC Score, and F1 Score obtained from the test

	precision	recall	f1-score	support
0.0	1.00	1.00	1.00	331
1.0	1.00	1.00	1.00	432
2.0	0.99	1.00	1.00	310
3.0	1.00	1.00	1.00	245
4.0	1.00	1.00	1.00	498
5.0	1.00	1.00	1.00	247
6.0	1.00	1.00	1.00	348
7.0	1.00	0.99	1.00	436
8.0	1.00	1.00	1.00	288
10.0	1.00	1.00	1.00	331
11.0	1.00	1.00	1.00	209
12.0	1.00	1.00	1.00	394
13.0	1.00	1.00	1.00	291
14.0	1.00	0.99	0.99	246
15.0	1.00	1.00	1.00	347
16.0	1.00	1.00	1.00	164
17.0	1.00	0.64	0.78	144
18.0	0.99	1.00	0.99	246
19.0	1.00	0.94	0.97	248
20.0	0.79	1.00	0.88	266
21.0	1.00	0.95	0.97	346
22.0	1.00	1.00	1.00	206
23.0	0.95	1.00	0.97	267
24.0	1.00	1.00	1.00	332
accuracy			0.99	7172
macro avg	0.99	0.98	0.98	7172
weighted avg	0.99	0.99	0.99	7172

Accuracy: 98.71723368655884

Fig: Classification Report of CNN model

Along with the calculated performance measures as shown in table above, we also obtained a confusion matrix. A confusion matrix gives a depiction of the true positives and negatives obtained after running the model. This gives a better understanding as to where the model was faulty and how many true negatives or false positives are there. The confusion matrix helps in deducing that most of the predictions made by the model are correct. However, some errors can be suspected due to the similarity in some letters. For instance, the fingers' positioning is very similar for O and S, which leads to some faulty predictions. Since we are using the MNIST dataset, most of the predictions are correct. Some of the predictions of our network

- **For ANN model:**
  After training the model, it was tested on testing dataset. We have used performance measures like accuracy, precision, recall and f1 score. The results are shown in table1:

	Accuracy	Precision	Recall	AUC Score	F1 Score
Result	0.8353	0.83	0.82	0.9142	0.82

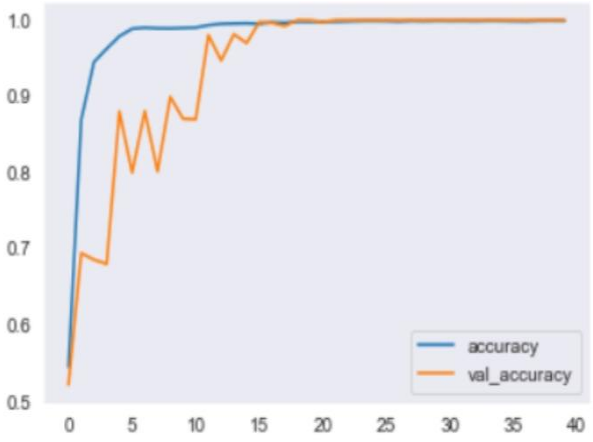


Fig: Accuracy Evalution

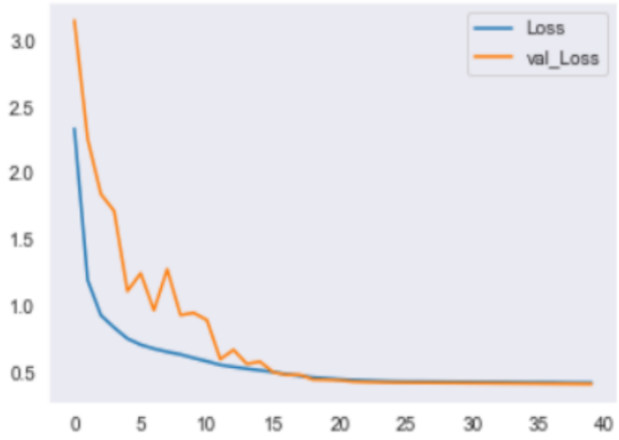


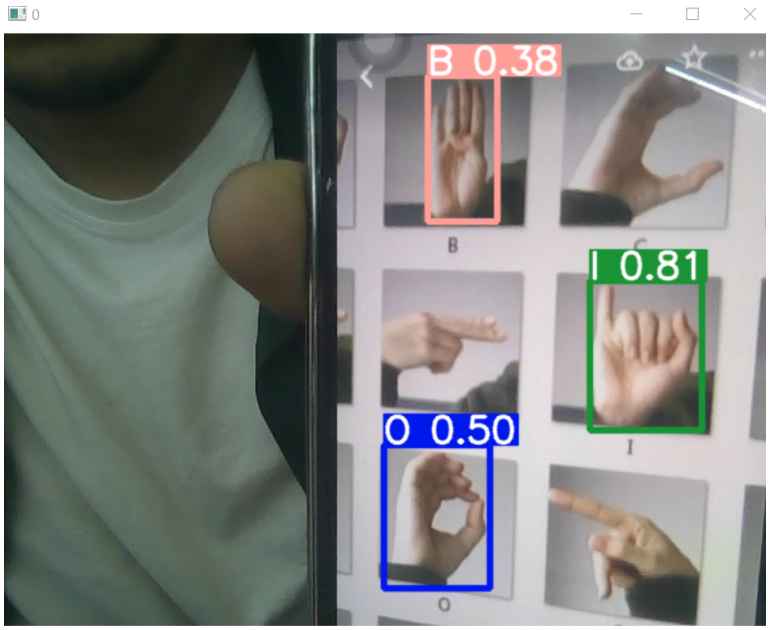
Fig: Loss Evalution

We trained our model on 20 epochs with a batch size of 512. The maximum accuracy obtained on the training set was 83.44%%, and the invalidating set was 85% for epochs. The accuracy graph is shown for training and validation accuracy, training and validation loss, respectively.

	precision	recall	f1-score	support
0.0	0.78	1.00	0.88	331
1.0	1.00	0.95	0.97	432
2.0	1.00	1.00	1.00	310
3.0	0.95	1.00	0.97	245
4.0	0.92	0.96	0.94	498
5.0	0.92	1.00	0.96	247
6.0	0.87	0.93	0.90	348
7.0	0.93	0.89	0.91	436
8.0	0.80	0.85	0.82	288
10.0	0.78	0.59	0.67	331
11.0	0.83	1.00	0.90	209
12.0	0.85	0.89	0.87	394
13.0	0.88	0.51	0.65	291
14.0	1.00	0.92	0.96	246
15.0	0.96	0.99	0.98	347
16.0	0.72	0.98	0.83	164
17.0	0.32	0.69	0.44	144
18.0	0.81	0.67	0.73	246
19.0	0.68	0.68	0.68	248
20.0	0.63	0.58	0.60	266
21.0	1.00	0.60	0.75	346
22.0	0.65	0.83	0.73	206
23.0	0.78	0.83	0.80	267
24.0	0.81	0.57	0.67	332
accuracy			0.83	7172
macro avg	0.83	0.83	0.82	7172
weighted avg	0.85	0.83	0.83	7172

Accuracy: 83.4495259341885

Figure: The classification report for ANN model



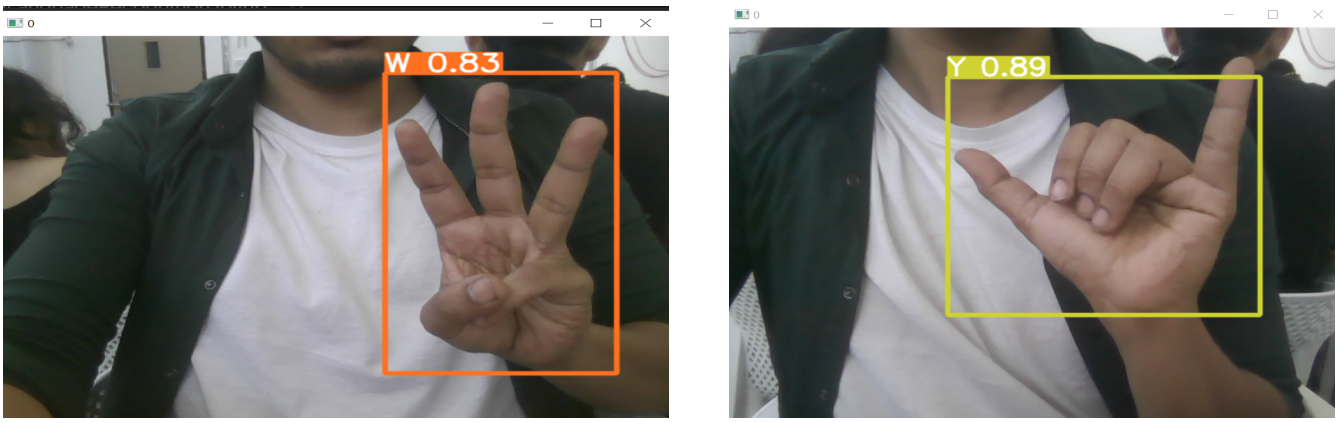


Fig: some pictures of our trained model detecting the alphabets

Conclusion and future scope :

If you have a deaf or hard-of-hearing friend, sign language is the only way to communicate with them effectively. People all over the world are starting to use this type of sign language to communicate effectively with the deaf these days. It is possibly the most effective way of communication for the deaf. Learning American sign language translation, on the other hand, may be difficult due to the fact that it is a motor rather than a cognitive skill. We used various techniques to remove overfitting in our model which include Batch Normalization, dropout layers, Regularization, and early stopping. The proposed Model was able to achieve an accuracy of 92% on our test dataset. The goal of our project is to create a sign language for text translators to solve this problem.

For the future perspective, we can also try to measure the accuracy of this model with different TensorFlow libraries and compare their accuracy among these models. In this way, we allow a new field to broaden up as then the accuracy of the sign language recognition will be effective.

References:

[1] S. Y. Kim, H. G. Han, J. W. Kim, S. Lee, and T. W. Kim, “A hand gesture recognition sensor using reflected impulses,” IEEE Sens. J., vol. 17, no. 10, 2975–2976,2017,doi: 10.1109/JSEN.2017.2679220.

[2] Y. Cui and J. Weng, “A learning-based prediction-and-verification segmentation scheme for hand sign image sequence,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 21, no. 8, pp. 798–804, 1999, doi: 10.1109/34.784311.

[3] A. Kumar and A. Kumar, “Dog Breed Classifier for Facial Recognition using Convolutional Neural Networks,” pp. 508–513, 2020.

[4] K. L. Bouman, G. Abdollahian, M. Boutin, and E. J. Delp, “A low complexity sign detection and text localization method for mobile applications,” IEEE Trans. Multimed., vol. 13, no. 5, pp. 922–934, 2011, doi: 10.1109/TMM.2011.2154317.

[5] J. Wu, L. Sun, and R. Jafari, “A Wearable System for Recognizing American Sign Language in Real-Time Using IMU and Surface EMG Sensors,” IEEE J. Biomed. Heal. Informatics, vol. 20, no. 5, pp. 1281–1290, 2016, doi: 10.1109/JBHI.2016.2598302.

[6] C. Zhang, W. Ding, G. Peng, F. Fu, and W. Wang, “Street View Text Recognition With Deep Learning for Urban Scene Understanding in Intelligent Transportation Systems,” IEEE Trans. Intell. Transp. Syst., pp. 1–17, 2020, doi: 10.1109/tits.2020.3017632.

[7] M. Safeel, T. Sukumar, K. S. Shashank, M. D. Arman, R. Shashidhar, and S. B. Puneeth, “Sign Language Recognition Techniques- A Review,” 2020 IEEE Int. Conf. Innov. Technol. INOCON 2020,1–9,2020,doi:10.1109/INOCON50539.2020.9298376.