Hand Gesture Recognition through Background Elimination

Final Project Report

Tanuj Sood
tanuj.sood_ug21

Honey Khandelwal
hitesh.khandelwal_ug21

Neeraj Pandey
neeraj.pandey_ug21

Abstract

This project uses convolutional neural networks to recognise hand gestures through a user's webcam in a specified region of interest. We explore the feasibility of recognising hand gestures in real time and identifying them with a high level of confidence. Our project began with attempting to recognise sign language in real time and converting it to speech to aid people with disabilities understand text. Though we had high training and validation scores, our real time prediction scores were very low. We tried to tackle this by creating our own dataset which had similar results. Our final idea was to use background elimination to get rid of the noise in our real time data and recognise hand gestures in real time through a webcam. Thus, using this technique our results yielded high real time prediction scores when trained through a CNN model.

1. Introduction

As our technology has advanced with rapid breakthroughs in human-computer interaction, hand gesture recognition has become a significant problem. Owing to this innovation, gesture recognition has been required by new HCI methods that allow us to interact with machines seamlessly with a higher level of ease. Be it sign language recognition which can help disabled persons browse through the internet or gesture sentiment recognition which can predict a human being's sentiments through their hand gestures, this technology has a great value in upcoming technologies in fields such as augmented reality (AR), robot control etc.

With a wide range of applications and practicality in the real world, this projected invigorated our interest and we started experimenting with different ways to read in images, train our dataset and then be used in a usable way to predict hand gestures. In our project, we make use of different models to predict hand gestures in real time with high accuracy.

2. Problem Definition

Input: Real-time continuous image sequences taken from the webcam.

Output: Prediction of hand gesture label with confidence.

Class Labels:

Class 0: Fist

Class 1: Hi

Class 2: Peace

3. Related Work

Gesture Recognition models have existed for quite some time now. The latest advancement in this field was in 2020 using CNNs [1]. The paper explores an efficient way to filter out hands from images through image masking, finger segmentation, normalisation of segmented images and recognition through the CNN classifier.

Another research paper [2] introduces PointLSTMs with the deployment of point clouds and LSTMs to outperform previous skeleton-based methods on the most challenging datasets NVGesture and SHREC'17. Though these powerful models exist, we will implement background image in a specified region of interest and separate the input from the surroundings by masking the image. We will see if we are able to compare to these state-of-the-art algorithms.

4. Dataset and Features

4.1 Sign Language Recognition Dataset

Our initial dataset was obtained from a sign language repository from Kaggle with twenty-six class labels representing twenty-six letters in the alphabet and 27455 images in total which is then converted to grayscale during preprocessing. For each image, the first column represented the class label of the image along with 784 features where each column gave us pixel data. Each image is of dimension 28x28 which after flattening gives us 784 features. Therefore, the dataset has the dimensions (number of images,

features) = (27455, 784). The grayscale colour of the image was used to quantify the pixel data for each pixels in term of darkness.

Fig 1. Images of each class label in the Kaggle dataset.

4.2 Custom Grayscale Sign Language Dataset

In our second approach, we made an attempt to create our own dataset due to certain limitations of the first dataset mentioned later in this paper. In this approach, we obtained data for five of the class labels (A, B, R, Y, W) in the sign language dataset by clicking pictures through our webcam and converting them to grayscale for easier classification. We collected five hundred images for each of the five chosen class labels and trained our model on these images with the hope of achieving better real time results. A few samples of the collected data have been given below.

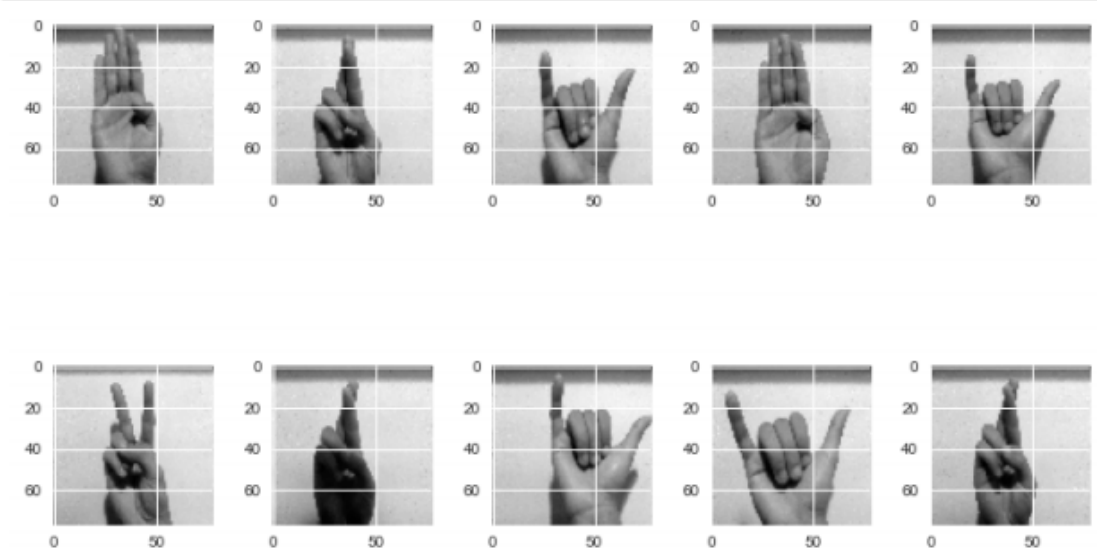


Fig 2. Self-created sign language dataset using laptop webcam.

4.3 Background Elimination Hand Gesture Dataset

Our final approach was to create another dataset, but with the use of background elimination algorithm to reduce background noise in our data and make it easier to classify with the presence of only two colours in the dataset. Due to computational restrictions, we couldn't train sign language models with 26 classes. Hence, we pivoted to recognising simple hand gestures and then proceeded to create our own dataset with three class labels. We had a total of six thousand training images with two thousand images for each class label. We had a total of three thousand testing images, with a thousand images for each class label. A few samples images of the dataset are given below.

Fig 3. Self-created hand gesture dataset using laptop webcam and background elimination.

Thus, we will now discuss our different approaches using these three datasets and how we used different methods and models to achieve a higher accuracy.

5. Methods

Our approach for all of our methods was to start with basic models, observe the accuracy we achieved and then move to better algorithms to acquire higher accuracy levels while experimenting with our dataset and pre-processing techniques to try and maximise our final results. This allowed us to analyse the setbacks of each method and gave us a deeper understanding of training models for real time prediction.

5.1 Sign Language Recognition on Standard Dataset

The initial aim of our project was to recognise sign languages and convert the identified signs to speech using CNN or MLP models which

denote what letter of the alphabet a given sign represents using text-to-speech algorithms.

Given an input image from the Kaggle dataset [3], we decompose it to separate the given class label of the image from the pixel data. Then, we normalise and standardise the pixel data. To perform an efficient training cycle, we split the data into three parts, 70% training split, 15% validation split and 15% testing split. and then found our loss curve and accuracy from the validation split.

5.1.1 Model Training

Model Preprocessing: The range of values of image pixels were in range (0, 255) before preprocessing. Preprocessing techniques like normalisation and standardisation was performed on this data to centre data points around the mean, reduce the variance and to ensure that data is internally consistent.

Model: Two models were used for training as mentioned below.

5.1.2 MLP Model

Model Architecture (MLP1):

Layers	Number of Neurons	Activation
Input Layer	784	None
Hidden Layer 1	50	Relu
Hidden Layer 2	50	Relu
Hidden Layer 3	75	Relu
Hidden Layer 4	50	Relu
Output	3	SoftMax

After preprocessing the data, the training model used was a multilayer perceptron algorithm (MLP) with model architecture as shown above. L2 regularisation was used to avoid the problem of overfitting. Dynamic learning rate was used to dynamically reduce the learning rate as the optimiser converge closer to the minimum and this was provided by TensorFlow's package called ReduceLROnPlateau.

Model Results:

Model Name	Training Accuracy	Validation Accuracy	Testing Accuracy	Real Time Accuracy
MLP1 + Adam	99%	94.5	96.3%	12%
MLP1 + SGD	82%	74%	77%	10%

The learning curve for this model has been given below.

Fig 4. Learning Curve of the Sign Language Recognition MLP model.

Limitations: Though the obtained accuracy of this model was extremely high, while testing it real time using a webcam, the performance of the model was with a 12% accuracy level over a hundred trials. This accuracy was extremely low and unanticipated after achieving high scores during validation and testing over the dataset.

5.1.3 CNN Model

Model Architecture (CNN-Sequential1):

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 28, 28, 75)        750
_________________________________________________________________
batch_normalization (BatchNo (None, 28, 28, 75)        300
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 14, 14, 75)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 14, 14, 50)        33800
_________________________________________________________________
dropout (Dropout)            (None, 14, 14, 50)        0
_________________________________________________________________
batch_normalization_1 (Batch (None, 14, 14, 50)        200
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 7, 7, 50)          0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 7, 7, 25)          11275
_________________________________________________________________
batch_normalization_2 (Batch (None, 7, 7, 25)          100
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 25)          0
_________________________________________________________________
flatten (Flatten)            (None, 400)               0
_________________________________________________________________
dense (Dense)                (None, 512)               205312
_________________________________________________________________
dropout_1 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_1 (Dense)              (None, 24)                12312
=================================================================
```

After preprocessing the data, the training model used was a CNN with model architecture as shown above. Dropout was used to avoid the problem of overfitting.

Model Results:

Model Name	Training Accuracy	Validation Accuracy	Testing Accuracy	Real Time Accuracy
CNN1 + Adam + Dropout	99%	94.5	99.3%	14%
CNN1 + SGD	66%	56%	52%	6%
CNN1 + SGD + Dropout	82%	74%	77%	10%

The learning rate of this model has been depicted through the increase in accuracy levels and the decrease of our loss value in the first ten epochs using the graphs below. Dynamic learning rate was used which is provided by TensorFlow's package called ReduceLROnPlateau.

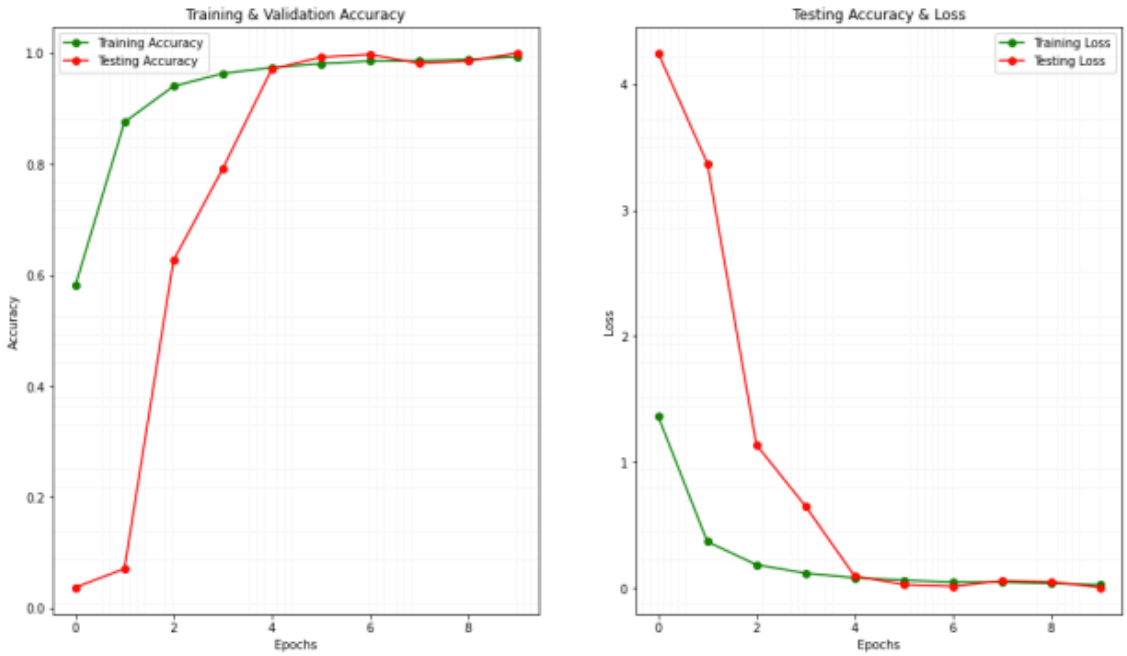


Fig 5. Learning Curve of a Sign Language Recognition CNN model.

Limitations: Acquiring higher results than the MLP model, the model was then tested on real time data. Though this model definitely

outperformed the MLP model over the dataset, real time accuracy were similar with 14% accuracy over a hundred trials. After further analysis, it was evident that the distribution of pixel data coming from the dataset was completely different than that of what the webcam was capturing which resulted in low real time accuracy. This conclusion led us to our second approach where we created and used our own dataset to train and test the model.

5.2 Sign Language Recognition on Self-Created Dataset

5.2.1 Model Training

Model Preprocessing: Raw images from webcam were captured which were then converted to grayscale. The range of values of image pixels were in range (0, 255) before preprocessing. Preprocessing techniques like normalisation and standardisation was performed on this data to centre data points around the mean, reduce the variance and to ensure that data is internally consistent. For details on the dataset, refer to section 4.1.

Model: Similar to Section 5.1, the same two models with same architectures were used for training on a custom dataset and the following results were obtained.

Model Results:

Model Name	Training Accuracy	Validation Accuracy	Testing Accuracy	Real Time Accuracy
MLP1 + Adam	95	10	30	0%
MLP1 + SGD	82%	11%	20%	0%
CNN1 + Adam + Dropout	99%	21	18	2%
CNN1 + SGD	67%	18%	11%	2%
CNN1 + SGD + Dropout	72%	19%	11%	0%

5.2.2 Model Analysis

The loss curve for the MLP Model has been given in Fig 7 below.

Fig 7. Loss Curve of MLP Model with custom dataset

Using the CNN model on this custom data, the dropout layer values were initialised as 0.25 and 0.75 so as to avoid overfitting. Fig 8 depicts the graph of the training and validation accuracy as well as the loss curve for training and validation loss.

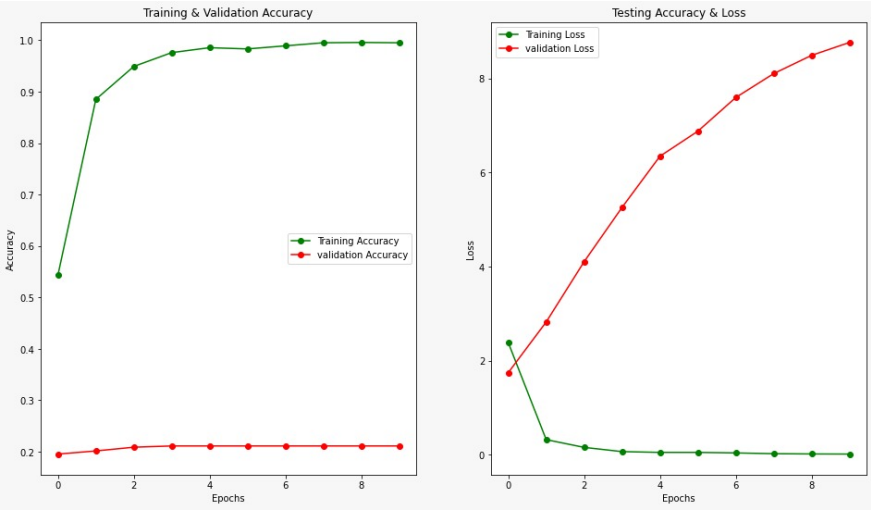


Fig 8. Accuracy and Loss Curves for the CNN model used with the custom dataset.

Using the graph above, we can observe that the model overfits the data because the training accuracy is high but validation and testing accuracy is low on both the models.

We conducted further research to come to a conclusion that the reasons why these models were inefficient had to do something with the data. Since the background of the image taken in real time varies significantly compared to the input dataset, there was a need to stabilise or remove the background and reduce background noise. Therefore, our next step was to implement background elimination algorithm on the dataset.

Since sign language data had 26 classes, to avoid overfitting we would need at least 10 lakh image data and due to computational restrictions, we were unable to train on this large dataset. Hence, we pivoted to creating a simpler dataset, i.e. simple hand gesture with three class labels. For more details on this data, please refer to Section 4.3 of this report.

5.3 Hand Gesture Recognition Using Background Elimination

5.3.1 Data

To know about the data, refer to section 4.3.

5.3.2 Data Pre-processing

Unlike the preprocessing data done in previous models, here while creating a custom dataset through webcam, instead of using the raw grayscale image as input, we implement background subtraction algorithm on the input data and then pass this image data forward for training.

5.3.3 Background subtraction algorithm

Background subtraction Algorithm is used to separate elements in the foreground from the background and this is achieved by generating a foreground mask. One uses a background subtraction algorithm to detect dynamically moving objects in an image sequence (video).

Step 1: The video sequence captured from the webcam is analyzed over a particular set of frames, in our case we have set it 30.

Step 2 : We then take the running average of current and previous frames in this 30 seconds window and then set this as the background for next 30 seconds window

Step 3 : Any new object introduced during this 30 second window,becomes the part of foreground. Therefore, now we have an estimated background (running avg) and a foreground element.

Step 4: Now we take the absolute difference between the estimated background and the current frame and if the difference is greater than a threshold we assign that pixel value as white else black

Step 5: We then find the contours based on the light intensity of the pixel. Since all the foreground, in our case hand gesture, will have white pixel value (because of step 4), we will get this hand gesture region/segment in contours and then we send this region to our training model or as input to make real time predictions.

5.3.4 Model Architecture

5.3.4.1 Baseline MLP3 Model

Layers	Number of Neurons	Activation
Input Layer	51600 (240x215)	None
Hidden Layer 1	75	Relu
Hidden Layer 2	75	Relu
Hidden Layer 3	75	Relu
Hidden Layer 4	50	Relu
Hidden Layer 5	25	Relu
Hidden Layer 6	25	SoftMax
Output	3	SoftMax

5.3.4.2 CNN3 Model Architecture

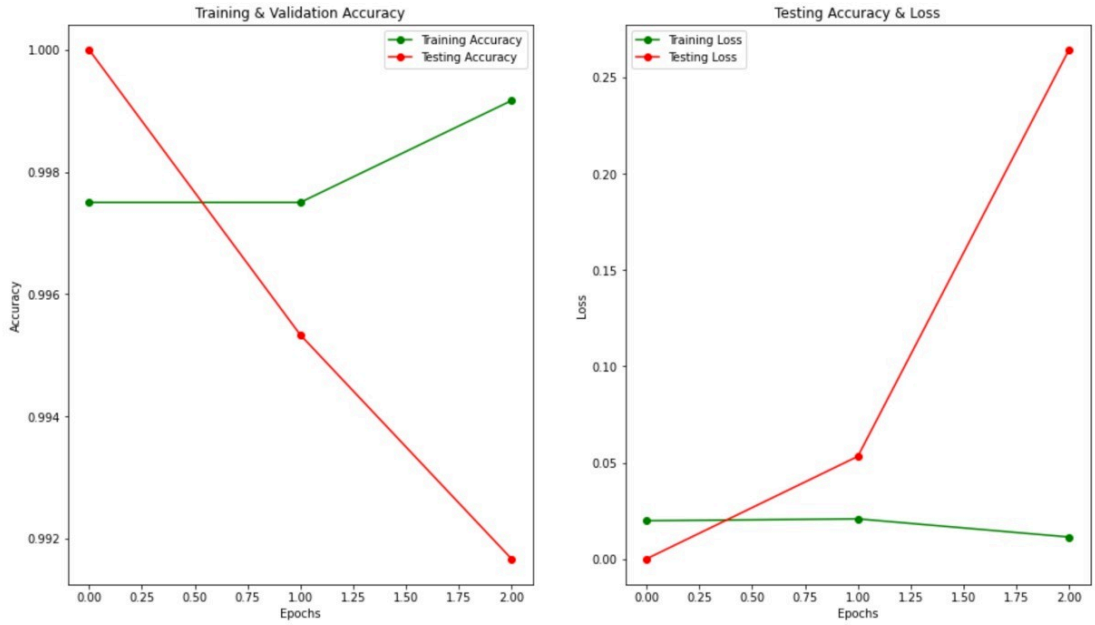
```
Model: "sequential_4"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_12 (Conv2D)           (None, 240, 215, 75)      750
_________________________________________________________________
batch_normalization_12 (Batc (None, 240, 215, 75)      300
_________________________________________________________________
max_pooling2d_12 (MaxPooling (None, 120, 108, 75)      0
_________________________________________________________________
conv2d_13 (Conv2D)           (None, 120, 108, 50)      33800
_________________________________________________________________
dropout_8 (Dropout)          (None, 120, 108, 50)      0
_________________________________________________________________
batch_normalization_13 (Batc (None, 120, 108, 50)      200
_________________________________________________________________
max_pooling2d_13 (MaxPooling (None, 60, 54, 50)        0
_________________________________________________________________
conv2d_14 (Conv2D)           (None, 60, 54, 25)        11275
_________________________________________________________________
batch_normalization_14 (Batc (None, 60, 54, 25)        100
_________________________________________________________________
max_pooling2d_14 (MaxPooling (None, 30, 27, 25)        0
_________________________________________________________________
flatten_4 (Flatten)          (None, 20250)             0
_________________________________________________________________
dense_8 (Dense)              (None, 512)               10368512
_________________________________________________________________
dropout_9 (Dropout)          (None, 512)               0
_________________________________________________________________
dense_9 (Dense)              (None, 3)                 1539
=================================================================
```

5.3.5 Model Result

Model Name	Training Accuracy	Validation Accuracy	Testing Accuracy	Real Time Accuracy
MLP3 + SGD	67%	30%	21%	5%
CNN3 + Adam + Dropout	99%	99%	99%	100%
CNN3 + SGD + Dropout	35%	23%	21%	5%

5.3.6 Loss and Accuracy Curves

CNN3 + Adam + Dropout (Best Model so far)



6. Results and Discussion

After a series unsuccessful attempts, we analysed the errors made with each step and were able to find ways to enumerate those mistakes and get an effective model by the end of our test. Our main quantitative metric which we used to analyse the success of each of our methods was the accuracy levels and While testing the model on real time data, we went from receiving 16% accuracy to 99% accuracy over a hundred trials and were able to classify each of the three class labels perfectly with a 99% confidence level. The only limitation we faced while trying to come up with a realistic model was our computational power since the CNN model took over 16 hours to train with a high accuracy on Google Colab. The figures below show the output of the real time tests.

However, even though we achieved an extremely accurate model, this does not go on to say that we performed better than the state-of-the-art Finger Segmentation Model[1] and PointLSTM model [2] which received real-time accuracy levels between 86% - 92%. This is as our model performed on a small scale with only three class labels and relatively less noisy backgrounds.

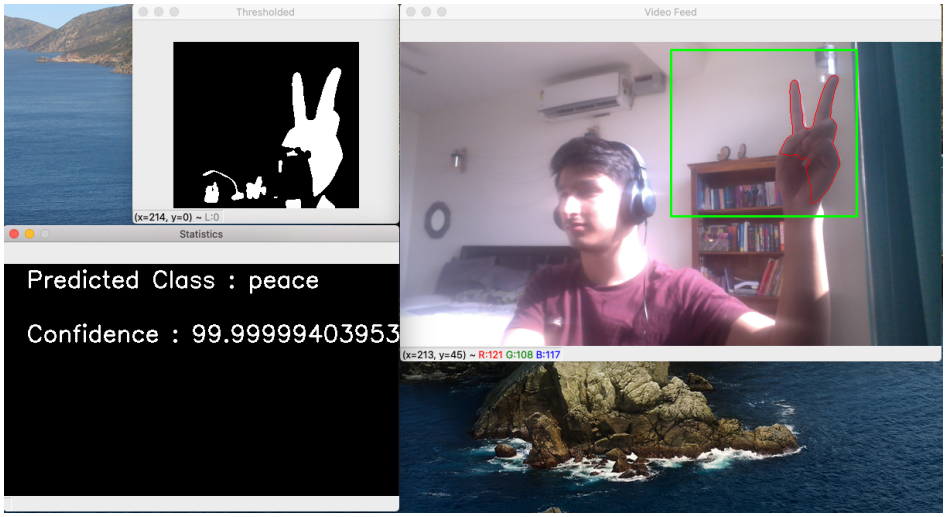


Fig. Prediction of Class Label “Peace” using background elimination

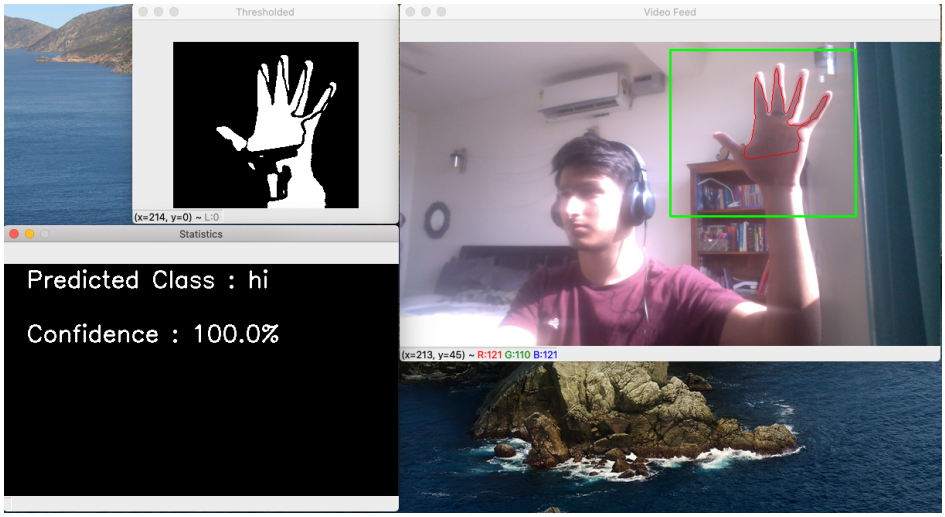


Fig. Prediction of Class Label “Peace” using background elimination

At the same time, these models have been trained on large datasets and can recognise a wide number of class labels despite of the noise in the background. Therefore, though we perform really well given a particular background and a small number of class labels, our machine learning model will most likely not surpass the latest models if we perform on the same training datasets.

7. Conclusion and Future Work

Comparison with state of art algorithms : State of art algorithms like .. have accuracy around 84-86 whereas our algorithm has accuracy well over 98% but this does not imply that our model is the new state of art because

this project is a small scale project when compared to large scale dataset that the state of art models used. Therefore, at this stage we cannot make a comparison with the state of art results but we wouldn't be surprised if in future, after scaling this dataset or using a benchmark dataset we beat the state of art results. Background subtraction is just a fundamental building block, in future we plan to incorporate RNN's and LSTM on top of this.

GitHub Link: https://github.com/neerajp99/hand_gesture_background

8. References

[1] Neethu, P.S., Suguna, R. & Sathish, D. An efficient method for human hand gesture detection and recognition using deep learning convolutional neural networks. *Soft Comput* **24,** 15239–15248 (2020). https://doi.org/10.1007/s00500-020-04860-5

[2] An Efficient PointLSTM for Point Clouds Based Gesture Recognition. (n.d.). Retrieved December 06, 2020, from https://paperswithcode.com/paper/an-efficient-pointlstm-for-point-clouds-based

[3] https://www.kaggle.com/madz2000/cnn-using-keras-100-accuracy

[4] AlSaedi, Ahmed & Hanon AlAsadi, Abbas. (2020). A new hand gestures recognition system. Indonesian Journal of Electrical Engineering and Computer Science. 18. 49. 10.11591/ijeecs.v18.i1.pp49-55.

[5] Authentise.Com, 2020, https://www.authentise.com/post/how-to-track-objects-with-stationary-background.

[6] Khaled, Hazem & Sayed, Samir & Saad, El & Ali, Hossam. (2015). Hand Gesture Recognition Using Modified 1$ and Background Subtraction Algorithms. Mathematical Problems in Engineering. 2015. 1-8. 10.1155/2015/741068.

[7] "Background Subtraction In An Image Using Concept Of Running Average - Geeksforgeeks". Geeksforgeeks, 2020, https://www.geeksforgeeks.org/background-subtraction-in-an-image-using-concept-of-running-average/.

[8] Arxiv.Org, 2020, https://arxiv.org/ftp/arxiv/papers/1407/1407.4898.pdf.