Text Generation through Hand Gesture Recognition

[1]Uday Khati, [2]Prajitesh Singh, [3*]Achyut Shankar

[1,2,3*]Dept. Of Computer Science and Engg.
ASET Amity University, Sector-125 NOIDA,
INDIA
[1]uday.khati@student.amity.edu, [2]prajitesh.singh@gmail.com, [3*]ashankar1@amity.edu

Abstract: Development of Vision-based Hand Gesture Recognition systems has proved to be beneficial for disabled people over the past. These systems are user friendly and inexpensive. They also make room for contemporary endeavours in the Human-Computer Interaction area. Gestures can control the cursor, the music player, or even play games. Despite all the improvements made in Vision-based recognition systems, a fast, accurate, and reliable system is yet to see the light of the day. This study aims to understand the previous work in this field and develop a fast and accurate alternative to perform the same task. To render the system faster, we propose to replace the histograms from the previous approach and uses native OpenCV functions. The Use of these functions can reduce the time for calculating the histogram and loading it when required. The final system developed, succeeded in predicting 37 gestures with a 98% accuracy. The system also has a calculator mode, where some gestures are reserved for operators and others for operands. This study tells us how we can use different computer vision techniques to make the recognition process more memory efficient and lightweight. In the future, this could be used as a commercially viable option by putting it on app stores.

Keywords— deep learning, computer vision, hand gesture recognition, convolutional neural networks, thresholding, morphological closing.

1. Introduction

The area of research for this project is Hand Gesture Recognition, which is one of the topics resulting from recent advances in Computer Vision technology and Deep learning. We try to answer questions like, **How to put Neural Networks to work for predicting different Hand Gestures from real-time images? How to use computer vision to make the process of recognition faster and efficient?**

Work done previously in the area of Hand Gesture Recognition has given us hardware-dependent systems [1], [3] which use sensor-based hand gloves, armbands, wristbands. Such systems are expensive and complex to use. Then there are vision-based static gesture recognition software [2], [8] that use underlying computer vision technology. These software-based systems, without any requirements of gloves, etc. are inexpensive and very easy to use. Some systems are also capable of doing dynamic hand gesture recognition [4]. They use MHI (Motion History Images) for tracking the real-time motion of hands.

Different computer vision techniques are used to process images for feeding purposes. Some directly use resized images, but that is a memory and processing-intensive process as colored images take up memory and processing time. Some convert images to binary form, as it is easy to process and lightweight. There are different methods to perform this conversion. One system used Histogram [7] and Back-Projection for this purpose. However, that creates another overhead to calculate. Then we have to perform that overhead repeatedly in order to use the histogram.

So the existing methods are either both expensive and complex to use because of hardware involved in them, or they are dependent on lighting conditions and require a separate process to be performed or they have huge overheads, which make the program sluggish.

A better alternative is required to predict gestures faster and become independent of lighting conditions. Previous works say that OpenCV is a powerful tool for manipulating images. It has different functions that car replace histograms and give better results.

Therefore, this study tries to achieve faster and efficient processing of images, which in turn, will result in faster recognition. We will try to make various OpenCV functions like blurring, morphological corrections, and thresholding work. This will result in an improvement in the overall performance of the model.

If the functions mentioned above work out as expected, the dependence on histograms for changing lighting conditions will vanish. Besides, the time-consuming overhead will be removed. This paper will elaborate on a better way to recognize gestures, which is faster and robust than previous works. It also explains the functioning of Convolutional Neural Networks.

Main Contributions of the study to the Gesture Recognition field:

- A fast and more accurate method to recognize Hand Gestures using native OpenCV functions.
- A Deep Learning model capable of predicting ASL Gestures with 99.92% accuracy.

2. Literature Survey

2.1 History

The very first instance of gesture recognition by computers was produced by the father of Computer Graphics Ivan E. Sutherland in 1964 in which he used a light pen with a Sketchpad to interact with the computer. A user could draw a line by pressing a button on a light pen and pointing it on display. Since then, there has been a tremendous development in this field. Very advanced variants have emerged in the market since then.

2.2 Related Work

- *In 2016, K.A Bhaskaran, A.G. Nair ET. al.* [1] developed an Equipment based Gesture Recognition System using smart gloves to track and recognize Hand Gesture. It is an example of the systems that require external hardware to recognize hand gestures. Gloves have sensors attached to the gloves, which track hand/finger movement and transduce finger flexion into electric signals for determining hand posture. Although it is more accurate, it lacks the ease of use factor of simple Vision-based systems and also loses out on the cost factor of the whole system. The gloves contain flex sensors for gesture recognition. It converts Indian Sign Language into speech. Gloves' usability is not limited to gesture recognition, it is also useful in medical areas and video gaming.

- *In 2019 Hanwen Huang, Yanwen Chong et al.* [9] built a Vision-based Gesture Recognition system. The system proposed, detects skin based on a formula: R>85, R-B>10, R-G>10. After segmenting skin from the background, it converts the hand image into binary. Then applies pre-processing to fill the holes/cracks, using newly proposed improved total variation algorithm in the non-texture based method. After preprocessing, it extracts hand contours from the image. Since the image contains both face and hands, in order to extract only hand contours, it employs a 16 layer VGGNet for this arduous task. For Gesture Recognition from the extracted contours, it uses the Pyramidal-pooling module along with the attention mechanism. The original image is passed through a 3X3 convolution layer and a max_pool layer resulting in a ½-sized image of the original one. It achieves a 98.41% accuracy over a 900 images test dataset

- *In 2019, Chen Zhu et al.* [13] designed a vision based Hand Gesture Recognition based system, which uses Kinect Sensors to capture depth maps of Gestures. Then segmentation of 3D hand shapes from the background is done. The study proposes a new descriptor for pattern extraction of 3D shape features – 3D Shape Context. It provides information on each 3D point at multiple scales, as both local and global shape distribution are necessary for recognition. The resultant description of 3D points helps in constructing the final representation of hand gestures. Finally, it uses the time warping algorithm for recognizing gestures. Rigorous testing reveals that the developed system is robust to usual variations in lighting conditions, noise, or any transformations.

- *In 2018, Nuwan Munasinghe* [4] developed a dynamic hand gesture recognition system. This system recognizes moving gestures instead of static ones. It generates Motion History Images from consecutive frames using an algorithm measuring structural similarity. It also shows the use of various OpenCV functions like Gaussian Background Subtraction, median filtering, and Otsu's thresholding. After subtraction, we apply blur (kernel size 11) for smoothening. Otsu's thresholding changes the smoothened image into a binary image. Then it creates MHI from these binary images. The Neural Network classifies the Motion History Images and predicts the gestures.

2.3 Important concepts

Apart from these research papers, other important concepts required for a complete understanding of the project are Convolutional Neural Networks, Tensorflow/Keras, OpenCV, filtering, blurring, thresholding, morphological transformations, and histograms.

- *Convolutional Neural Networks*: They are an important type of deep neural networks. They are very powerful while working on the analysis of images, identifying patterns. Therefore, used in this project. The core idea of a Convnet is that it matches smaller parts or features of a test image with training data rather than the whole image. As the name suggests there is a window or a feature that convolutes over our image to check how well that filter matches that image. Convolution referring to the repeated application of the feature or filter over the whole image again and again for different positions in the image.

- *Keras*: Keras is a high-level API for Tensorflow, Theano, and other ML Libraries used for playing with deep neural networks. It makes experimenting with various combinations of parameters of a neural network very easy and helps in understanding core ML concepts. Keras is used in this project to create our deep learning model, which is a Convolutional Neural Network. It provides various types of layers like Conv2D, Maxpooling2D, Flatten, Dense, and Dropout. All these layers sequentially form our Neural Network.

  The best thing about Keras is that it makes core tasks of machine learning like building model and training it, so easy and convenient that we need to type just one line of code. Model.compile(), model.fit() and model.evaluate() are the three functions respectively for building, training and evaluating model (on validation data). We have some parameters to tweak to make the best model, like loss function, optimizer, metrics on which we will judge the model's performance.
  It also provides a high level, very useful functions like ModelCheckpoint(), to create a checkpoint during the building process of our model.

- *OpenCV*: It is a powerful computer vision library capable of performing very complex image operations smoothly. OpenCV has been at the heart of this project from the start. It has been put to use from the first task in the project, i.e. Image capture and processing. There on it is used everywhere, flipping the images horizontally, converting color spaces from BGR to Grayscale to HSV, etc. It is one of the main tools underlying the project.

- *Filtering, Blurring, Thresholding*: Filtering refers to filtering out some object from the image and keeping the rest. This is done with the help of a mask created with a specific color range of the object that we want to filter out from the image. cv2.inRange() function creates this mask using the lower and upper values provided corresponding to the color range. Finally, the bitwise_and() function of OpenCV filters out the mask from the image.

  Blurring refers to noise reduction. The image filtered out has a lot of noise in it. Such an image cannot be used for training the model. So we soften the image and remove noise from the image. Here we have used Median Blurring with a kernel size of 15x15 pixels.

  Converting a grayscale image into a black and white image (binary) is Thresholding. cv2.threshold() function is used for this purpose. We have various methods to perform thresholding using various flags. Some require an explicit threshold provided for thresholding (Threshold_Binary), others (Adaptive thresholding) have this ability to estimate an appropriate threshold for a particular small section of the image. The best method, which satisfies our purpose, is Otsu's Binarization. This method works only on a **bimodal image**, i.e. images which have only two colors, binary images. We have converted our image into a binary image using the thresholding function. Therefore, we can apply Otsu's method, which will automatically estimate a threshold value for our bimodal image.

- *Morphological Transformation (Closing)*: These transformations are performed on binary images. They are used to remove unwanted elements from the image and make our desired image more clear and in-focus. Basic transformations are Erosion, Dilation, Opening, and Closing. We have used Closing here, which is a combination of Erosion after Dilation. It means that first, we increase the white area of our image by dilating it and then remove the noise from the image and shred the boundaries of the foreground object which is in white (Hand in our case).

- *Numpy*: Numpy is the matrix/array manipulating library. Since the computer sees the images captured as arrays, it is a great tool to change the properties of images according to our needs. At many steps we need to reshape the image array, np.reshape() facilitates this. Np.zeroes(), np.ones() give us an array of 0s and 1s of desired dimensions, respectively.

- *Pyttsx3:* It is a python library used to convert text to speech, with advantages such as platform independence and offline support.

- *Os module:* It is a very useful tool to work with an operating system's file system. It is used to read/write files (read/write), get paths, and manipulate them (os.path), reading all lines of a file (fileinput), accessing environment variables (os.environ), etc.

- *Pickle:* It is a module used for storing (serializing) and loading (de-serializing) python objects. It serializes an object into byte streams for storing it on permanent memory, using the function pickle.dump(). While loading images, it retrieves bytes from the hard disk and brings it to the main memory to use in the program, through pickle.load() function.

- *Sqlite3:* A widely used database engine providing local storage on the user's device.

  The training dataset for the model is stored in this embedded database system called sqlite3. Written in C, it has very simple and easy commands to interact with the database. Initially, a *conn object* is created with sqlite3.connect() command. Then execute and commit() attributes are used to execute desired commands and save any changes to the database.

- *Matplotlib:* This library is used to plot different types of graphs in python programs. Our project used it to plot and check histograms of images and to plot confusion matrix.

3. Proposed Methodology

This study tries to solve the problem facing us using the functions of OpenCV and Deep Learning's Convolutional Neural Networks. OpenCV packs some very powerful functions inside it, which can fulfill our requirements quite well. Other than that, Convnets are the most powerful neural networks when it comes to image analysis and pattern recognition. Numpy library is a very useful tool for formatting images once converted into arrays, NumPy arrays.

Convnet takes processed images and respective labels as input and learns on its own during the training epochs/iterations. It checks its knowledge/learning through validation images and labels, which is a small part of data taken out of training data for evaluating the performance of the model.

3.1 *Phase I - Image Capturing (Pre-Training)*

Before feeding binary training images to the Neural Network, each image undergoes a long sequence of processing. These processes make images suitable for a Convnet to take as input so that the network can learn fast and accurately. Following steps summarize the operations performed on images:

- *Image Capture and Flipping:* Dell Inspiron 3542 laptop's webcam captures images at a resolution of 1280 x 720 at 20 fps. This resolution is later dropped down to 50 x 50. Captured images are randomly flipped for the sake of variation in the dataset. 800 images are captured. Then a flipped copy of each of these images is added to the dataset, resulting in a total of 1600 images for each gesture. This step makes the model more robust and prepares it for a left-hand user.

- *BGR (Blue Green Red) to HSV (Hue Saturation Value) conversion:* HSV color model is more accurate than the BGR. It portrays colors in the same way as humans see. BGR gives just a combination of three-color values, which is somewhat less on semantics. Hue gives the basic color, while Saturation gives brightness or tint to mix and Value indicates the amount of black or white present in the color. This mix of three variables truly represents the color according to the human world.

- *Filtering:* From the HSV images, skin-colored hand is filtered out with a *bitwise_and* function using a mask of color range HSV([0, 10, 60]) - ([20, 150, 255]). This step gives us a background-subtracted image containing skin-colored objects, hopefully, our hand only.

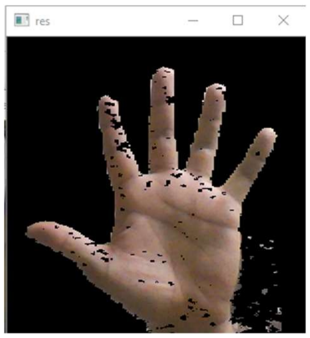


Fig. 1 Hand filtered out of the whole image, full of noise

- *Blurring:* Clearly, there is a lot of noise in the resultant image of the previous step. There is noise in the foreground (main subject) and in the background also. Median Blurring with a window size 15 is applied to reduce the noise. This makes the image very smooth and noise-free.

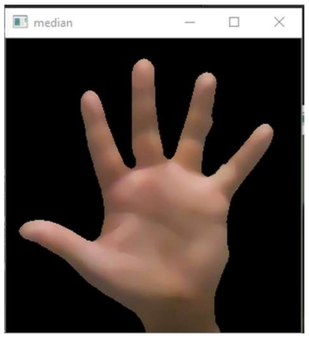


Fig 2. Smooth image after Median Blurring

- *Closing:* This step is responsible for closing out any of the holes that were left open in the image after the blurring process. Technically, it first performs dilation on the image and then does erosion on it. It removes any black points over the hand. A kernel of size 9 x 9 full of ones, is used for closing in on the holes of the hand.

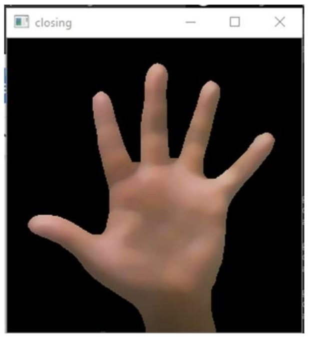


Fig 3. Solid Image free of any black holes

- *Thresholding:* After getting a smooth, solid image of the hand, we have to convert it from a colored image into a black and white binary image. Thresholding is an apt process for this task. However, thresholding requires the input image to be in grayscale mode. Since direct conversion from HSV to grayscale is not possible, we first convert the HSV image into BGR and then make it a grayscale image. The threshold for the grayscale image is kept 140 in the range of 0-255.

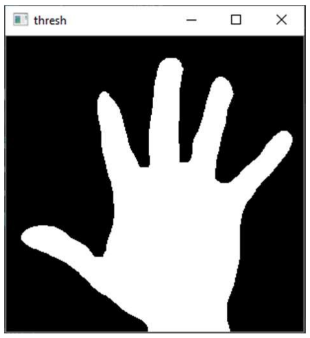


Fig 4. Binary image after Otsu's thresholding

Now the image is ready to be saved and loaded to model, only after resizing it to 50 x 50. However, if we resize the image above, we may lose out on some portion of the image. Therefore, to prevent any such case, we further reduce the dimensions of the image by *finding contours of image* and drawing a bounding rectangle just around the hand leaving out an unwanted black part of the image.

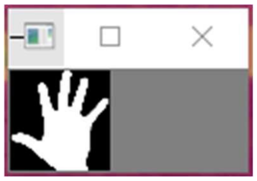


Fig 5. 50 x 50 image after bounding rectangle is drawn with the help of contours. Notice that the image ends with the white part of the image

3.2 Saving and Loading Dataset

Now this image is saved to the database and is ready to be loaded to the neural network. Like this, images of all the gestures are saved and loaded.

For saving and loading the images, python's pickle module is used. It is very useful for saving huge data sets or ML models from memory (RAM) to the hard disk. It serializes python objects into byte streams for storing on permanent memory and can also deserialize them when needed to run for the program. pickle.dump() is used while serializing or saving the data into the hard disk. It takes a total of 3 minutes to load the images. All gestures saved are shown in Fig 7 on the next page.

3.3 Building the Model

Now for training the model, high-level API Keras is used. It builds a Convolutional Neural Network model. Convnet is chosen because of its prowess in image analysis and picking up patterns in images. Three consecutive sets of *Covn2d layer and max_pooling2d* layer are used, apart from one initial conv2d layer for taking images as input. The final max_pooling2d layer of consecutive sets passes its output to a *flattening layer*. This layer flattens the output of the max_pooling2d layer into a single 1D vector. This is done because the forthcoming layers i.e. *Dense Layers* accept only 1D vectors/Tensors as input.

Deep Learning neural networks tend to overfit the model for the training dataset. Therefore, to avoid overfitting of our data a *Dropout Layer* is used for *reducing overfitting in the model.* Dropout refers to the random dropping of nodes from a layer, especially the hidden layer, to improve generalization. This process regularizes the large weights of a neural network, which reduces overfitting. Usually, a network's layers *co-adapt* to rectify errors coming from previous layers, which makes the model overfit to the training dataset. The Dropout layer does not allow such occurrences to happen, preparing the model for some unexpected situation.

Dense Layers are *fully connected* layers, which form the ending of our model and perform the final prediction.

3.4 Training the Model

The model trains over 20 epochs, with data being provided in a batch size of 500 images. In each epoch, the model first trains and then validates its learning by testing its learning on validation data provided. During the whole 20 epochs, the model adjusts and corrects *69,007 parameters*. Datasets are de-serialized into python objects again using *pickle.load()* function. Out of the total of 59,200 images, 49,333 images are kept for training while the remaining 9867 are further split into half as 4934 for validation and 4933 for testing. It takes a total of 22 minutes to train the model.

The trained Neural Network is retained on the disk for the prediction phase. This is also done using the pickle module. The final picture of the model is shown below:

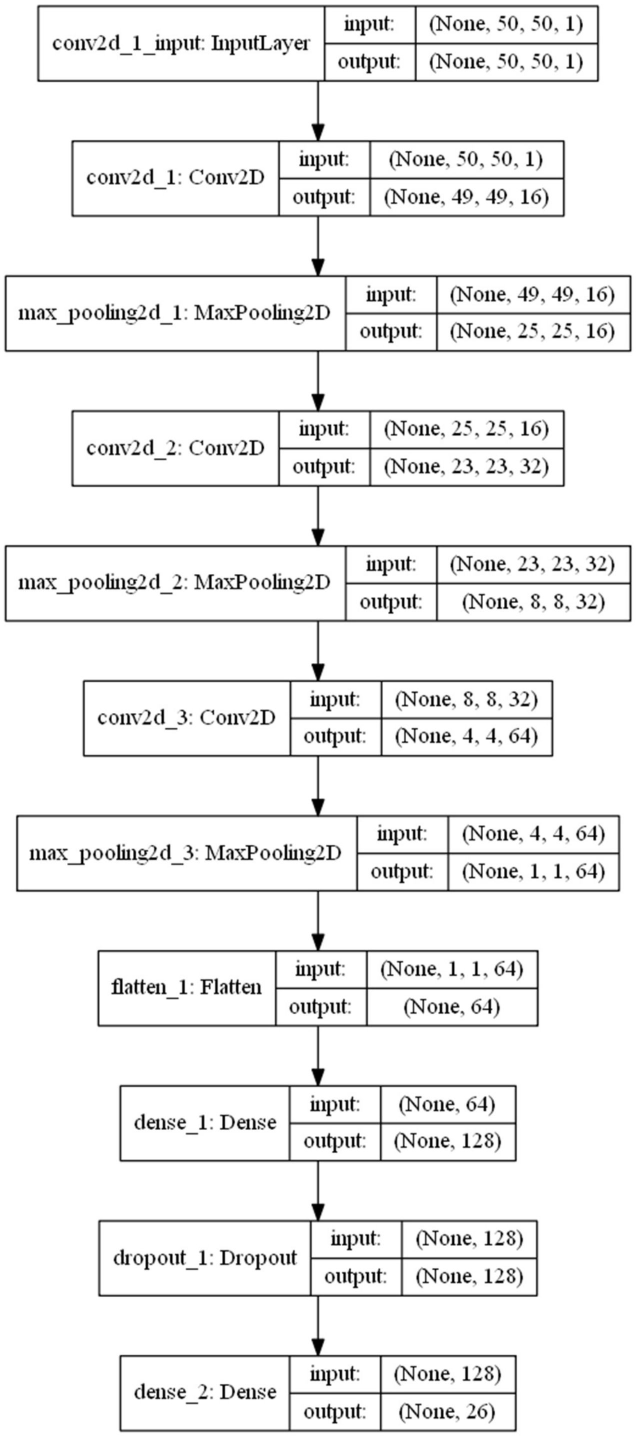


Fig 6. Layers in the model with their parameters

All the gestures stored in the database are shown below:

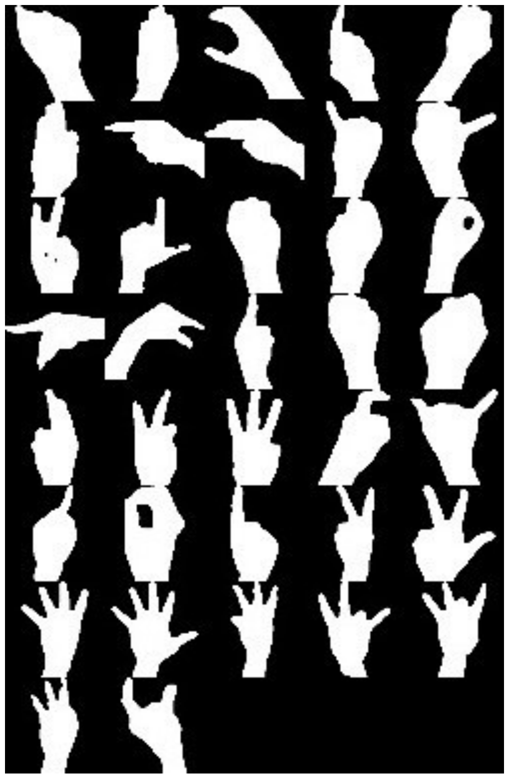


Fig 7. All gestures stored in the database

3.5 *Testing the Model*

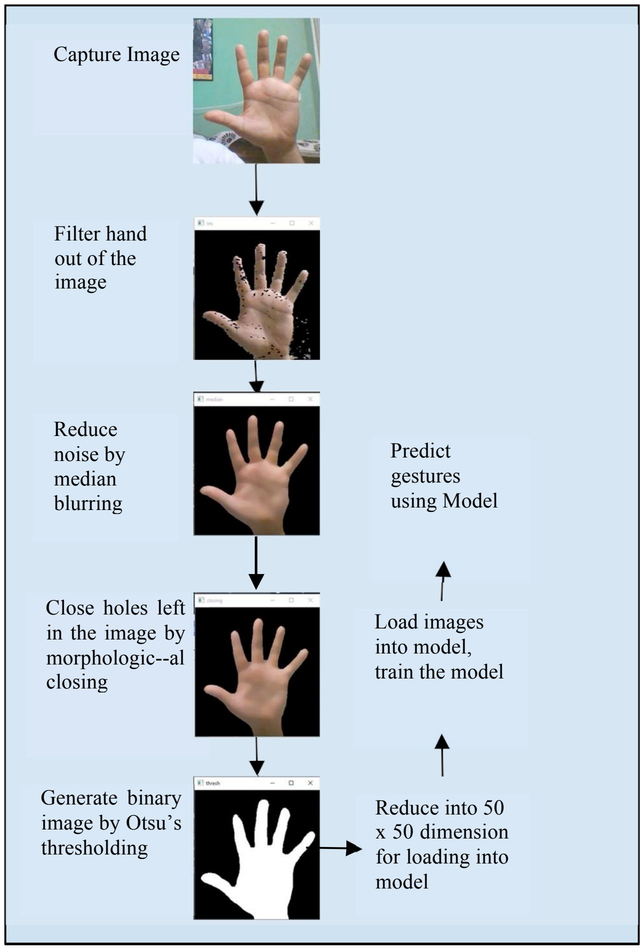
For verifying the accuracy in prediction of the model, fun_util.py is created. It has two modes in it namely, Text mode and Calculator mode. Text mode predicts the gesture, shows the text, and concatenates all the letters and once the hand is removed from the screen it says the resulting words/sentences out loud.

Text mode and Calculator mode both get predicted text corresponding to gesture shown, from the SQLite database. The Keras model is used to compute the probability of a gesture matching with some image stored in the database. If the probability comes to be greater than 0.7, then we fetch the text corresponding to the gesture from the SQLite database. The *Threading module* is used to concatenate the text together and form a word.

3.6 *Converting Text into Speech*

Python module *pyttsx* is used for generating speech from text predicted by the model. It is a cross-platform text to speech library. Another advantage is that it works offline for speech generation, which makes it more reliable.

4. System Architecture


Capture Image
Filter hand out of the image
Reduce noise by median blurring
Close holes left in the image by morphologic--al closing
Generate binary image by Otsu's thresholding
Reduce into 50 x 50 dimension for loading into model
Load images into model, train the model
Predict gestures using Model

5. Experiments and Analysis

This study required experimental work at multiple stages. At first, various methods (used in different research papers) were studied and pondered upon.

We tried working with the OpenCV Histogram method Saha, 2018 [8], but the histograms were an unnecessary overhead. It required to set a new histogram for the model, every time there was a change in lighting conditions. This was the biggest turndown when we compared it with a simple OpenCV functions based system. So we bent towards the process of filtering with a mask and then blurring. It was a fast and smooth method and gave even better results.

Experiment: For creating a mask according to Indian skin color, different values were tried. Finally [0, 10, 60] – [20, 150, 255] was finalized.
Analysis: In the end, it was realized that no range can be perfect, which would cover every section of the hand and leaves every undesired object in the image. In order to get a perfectly filtered image, we have to apply some extra functions like blurring and closing, etc.

Experiment: Different kernel sizes were also tried for blurring and closing separately. But Median Blurring worked best with the default *15*x*15 pixel window*.
Analysis: A smaller window would result in an insignificant change in the image i.e. noise in the image was not removed, while a larger than the 15-pixel window, smoothened the image to such an extent that it lost its details and wasn't of any use.

Experiment: For closing also different sized kernels were tried and a *9*x*9 kernel* was finalized to apply on the blurred image.

Analysis: A smaller kernel was not able to close the holes as the kernel would fit the hole itself and all the pixels inside the window would be 0, therefore, resulting in the same 0 intensity (black) hole. A much larger kernel would cover a large area and make the image fuzzier by filling the intentional gaps made by fingers too. So a perfect size kernel was required that would just fit our requirements. Below are the 2 images showing the disadvantages of using a larger kernel for closing.

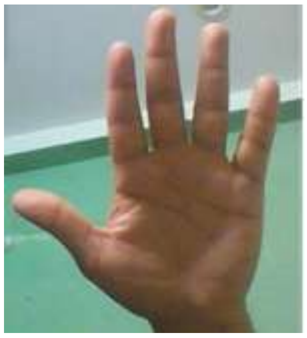


Fig. 9 Image of a Hand

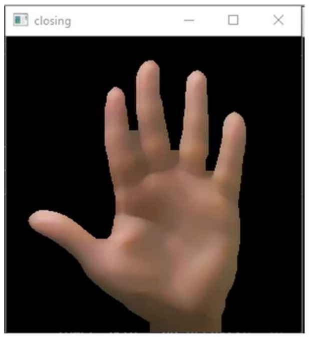


Fig. 10 Same image after filtering and closing with a large kernel

Notice that the **gap, between** the index finger and middle finger, is filled in the closed image while it is visible between the two fingers in the original image.

Here are some statistics showing the time taken by various parts of the system:

The time required for flipping 800 images on a vertical axis: **4min 27 sec**

The time required for loading 800 images: **3min 45sec**

The time required for training the Convnet (for 69,007 parameters): **22 min**

It was observed that the model was very sensitive to batch size. When it was changed from *500* images per batch to *330* images per batch, accuracy dropped from 99.92% to 72%.

Finally, the performance of the model is satisfactory with the given accuracy of 99.92%. Relative performance in comparison to previous approaches like sensor-based gloves, armbands it is very cheap and easy to use. It has lesser components to take care of. Its maintenance is also much affordable than a hardware-based system.

In comparison with other vision-based systems like those including histograms, it is found that it is far less bulky in terms of overheads and time consumption. We can directly use recognize.py to predict the gestures while otherwise we are required to set histogram first and then proceed further with prediction.

6. Results

At the end of the study, we have a deep learning model with us which is capable of predicting the probability of a gesture corresponding to a specific letter with an accuracy of 99.92%. The great thing about this system is that it can predict gestures with much ease than other vision-based systems using Histograms. It saves the memory as the histogram isn't required to be persisted to the disk (which is done using pickle), and one less file (which calculates histograms) is required.

The performance of the model is measured by some special parameters like **Precision**, **Recall,** and **f1-score**. In the figure displayed on the next page ***Classification Report****,* there we have shown all these metrics for the model. On testing the model based on these metrics, the average of Precision, Recall, and F1-score for all 37 gestures comes out to be 1.00 separately.

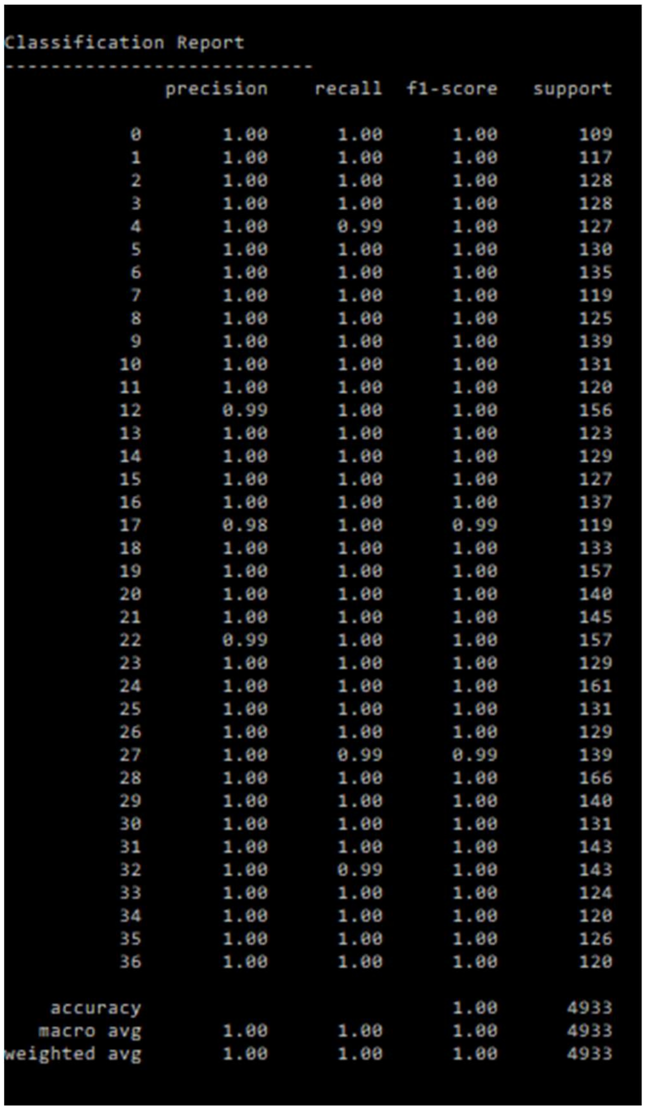
Precision which gives us a fraction of True Positives over Total Predicted Positive, is in a way an indicator of the accuracy of the model, telling us out of total gestures predicted positive for a letter, how many gestures were actually of that letter. It is a great tool to *reduce the number of false positives*. If a lot of false gestures are predicted to be of a letter, precision will drop greatly. But that is **not the case** in our model, as the precision is **100%**.

Then Recall is the ratio of True Positives over True Positive + False Negative i.e. True Positives over Total Actual Positives. It tells us how much percentage of our Actual Gestures (of a specific letter) are predicted as Positive i.e. to be of that letter. This helps us in reducing the percentage of False Negatives i.e. the no of actual gestures which weren't predicted to be of that letter. A large no of Gestures not reported as correct will *drop* the percentage in the recall.

In the context of our model, that is **not** much of **greater risk**. Because if we show a gesture and model doesn't recognize correctly ***for one frame or two frames***, it will not make a significant impact. The gesture will be recognized in subsequent frames correctly. Still, our model does well in this parameter also as the average of Recall for 37 gestures is also **100%**.

Finally, F1-score is the balancing parameter between Recall and Precision. It is better than plain accuracy because Accuracy can be driven by any type of correct predictions, worst case scenario being *huge cases of True Negatives* because they rarely matter to an organization. While F1-score takes into account the much more important False Positives and False Negatives. It also counters an uneven class distribution.

The F1-score of our model is also 100%. Such achievements make our model pass with flying colors.



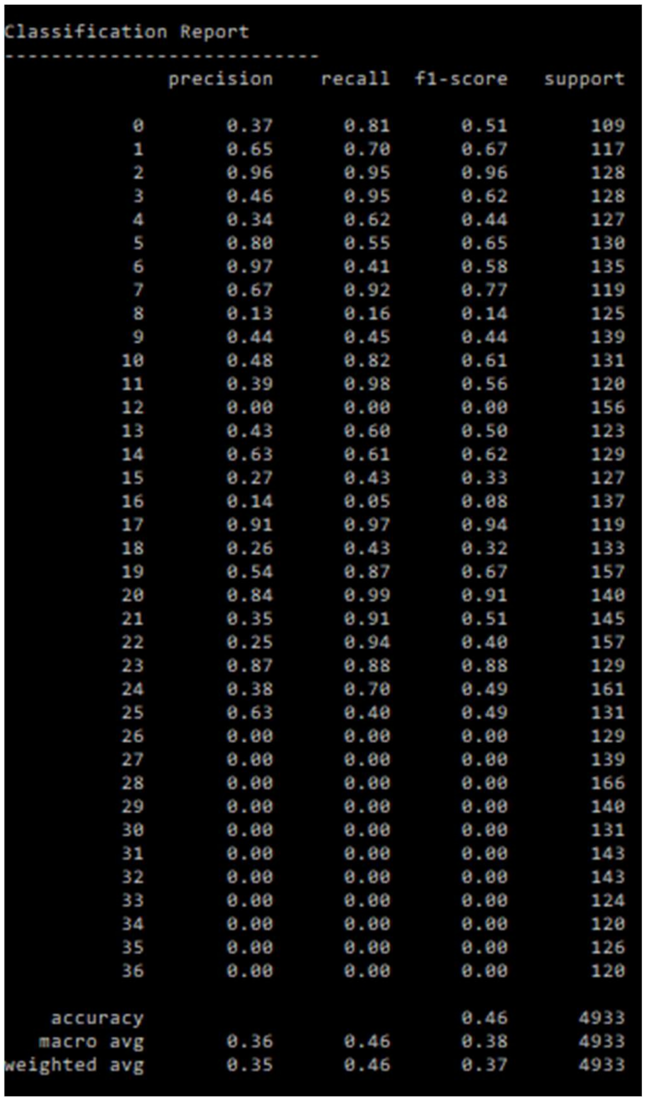
```
Classification Report
---------------------------
              precision    recall  f1-score   support

           0       1.00      1.00      1.00       109
           1       1.00      1.00      1.00       117
           2       1.00      1.00      1.00       128
           3       1.00      1.00      1.00       128
           4       1.00      0.99      1.00       127
           5       1.00      1.00      1.00       130
           6       1.00      1.00      1.00       135
           7       1.00      1.00      1.00       119
           8       1.00      1.00      1.00       125
           9       1.00      1.00      1.00       139
          10       1.00      1.00      1.00       131
          11       1.00      1.00      1.00       120
          12       0.99      1.00      1.00       156
          13       1.00      1.00      1.00       123
          14       1.00      1.00      1.00       129
          15       1.00      1.00      1.00       127
          16       1.00      1.00      1.00       137
          17       0.98      1.00      0.99       119
          18       1.00      1.00      1.00       133
          19       1.00      1.00      1.00       157
          20       1.00      1.00      1.00       140
          21       1.00      1.00      1.00       145
          22       0.99      1.00      1.00       157
          23       1.00      1.00      1.00       129
          24       1.00      1.00      1.00       161
          25       1.00      1.00      1.00       131
          26       1.00      1.00      1.00       129
          27       1.00      0.99      0.99       139
          28       1.00      1.00      1.00       166
          29       1.00      1.00      1.00       140
          30       1.00      1.00      1.00       131
          31       1.00      1.00      1.00       143
          32       1.00      0.99      1.00       143
          33       1.00      1.00      1.00       124
          34       1.00      1.00      1.00       120
          35       1.00      1.00      1.00       126
          36       1.00      1.00      1.00       120

    accuracy                           1.00      4933
   macro avg       1.00      1.00      1.00      4933
weighted avg       1.00      1.00      1.00      4933
```

Fig.11 Classification Report showing various metrics for the model's performance (batch size: 500)

The classification report shows the **individual** and **average value** of Precision, Recall, and f1-score for each gesture. As we can see, the weighted average of each of these metrics comes out to be 100%.



```
Classification Report
---------------------------
              precision    recall  f1-score   support

           0       0.37      0.81      0.51       109
           1       0.65      0.70      0.67       117
           2       0.96      0.95      0.96       128
           3       0.46      0.95      0.62       128
           4       0.34      0.62      0.44       127
           5       0.80      0.55      0.65       130
           6       0.97      0.41      0.58       135
           7       0.67      0.92      0.77       119
           8       0.13      0.16      0.14       125
           9       0.44      0.45      0.44       139
          10       0.48      0.82      0.61       131
          11       0.39      0.98      0.56       120
          12       0.00      0.00      0.00       156
          13       0.43      0.60      0.50       123
          14       0.63      0.61      0.62       129
          15       0.27      0.43      0.33       127
          16       0.14      0.05      0.08       137
          17       0.91      0.97      0.94       119
          18       0.26      0.43      0.32       133
          19       0.54      0.87      0.67       157
          20       0.84      0.99      0.91       140
          21       0.35      0.91      0.51       145
          22       0.25      0.94      0.40       157
          23       0.87      0.88      0.88       129
          24       0.38      0.70      0.49       161
          25       0.63      0.40      0.49       131
          26       0.00      0.00      0.00       129
          27       0.00      0.00      0.00       139
          28       0.00      0.00      0.00       166
          29       0.00      0.00      0.00       140
          30       0.00      0.00      0.00       131
          31       0.00      0.00      0.00       143
          32       0.00      0.00      0.00       143
          33       0.00      0.00      0.00       124
          34       0.00      0.00      0.00       120
          35       0.00      0.00      0.00       126
          36       0.00      0.00      0.00       120

    accuracy                           0.46      4933
   macro avg       0.36      0.46      0.38      4933
weighted avg       0.35      0.46      0.37      4933
```

Fig. 12 Classification Report showing various metrics for model's performance (batch size: 330)

The two juxtaposed classification reports (CR hereafter) of two models with different batch sizes (500 & 330) show the difference in performance caused by the batch size. Right CR depicts the poor performance of the model, with precision equal to 0.35, recall equal to 0.46, and f1-score being equal to 0.37. This teaches the importance of parameters like batch size, epochs, etc.

The next page shows the Confusion Matrix for the Deep Learning model, with batch size set to 500. Confusion Matrix plots on the graph, the Predicted labels against the True Labels. It shows how many times a gesture (on the X-axis) was predicted as some particular gesture (on the Y-axis).

E.g. if we see the 18th column (label no. 17) and 28th row (label no. 27) There is 2 instead of 0, meaning that the 17th label (gesture for R) was recognized as 27th label (gesture for digit 1). Moreover, at the intersection of the 18th column and 18th row, there is 119, meaning that the 17th label (gesture R) was recognized as the 17th label (R) 119 times.

Below the graph, it shows the accuracy of 99.92% and misclass of 0.8%.

On the next page is the confusion matrix for the Deep Learning model with batch size set to 330. That model churned out a poor accuracy of 46%. In that graph, at the intersection of 22nd column (label no 21) and 29th row (label no 28), there is 158, which means that 21st label (gesture for alphabet V) was recognized as 28th label (gesture for digit 2), 158 times during validation testing, which is a huge number. This is obvious though because gestures for V and 2 are pretty similar. But this issue was solved in the model with batch size 500.

Certain uncertainties are there regarding the lighting conditions of the environment. It is advised to run this system in a well-lit environment and must avoid a skin color background, as it will make it harder to extract the hand out of the background. Also if the system using this model has a GPU installed on it, it is advised to install the TensorFlow-GPU version to make the best out of GPU.

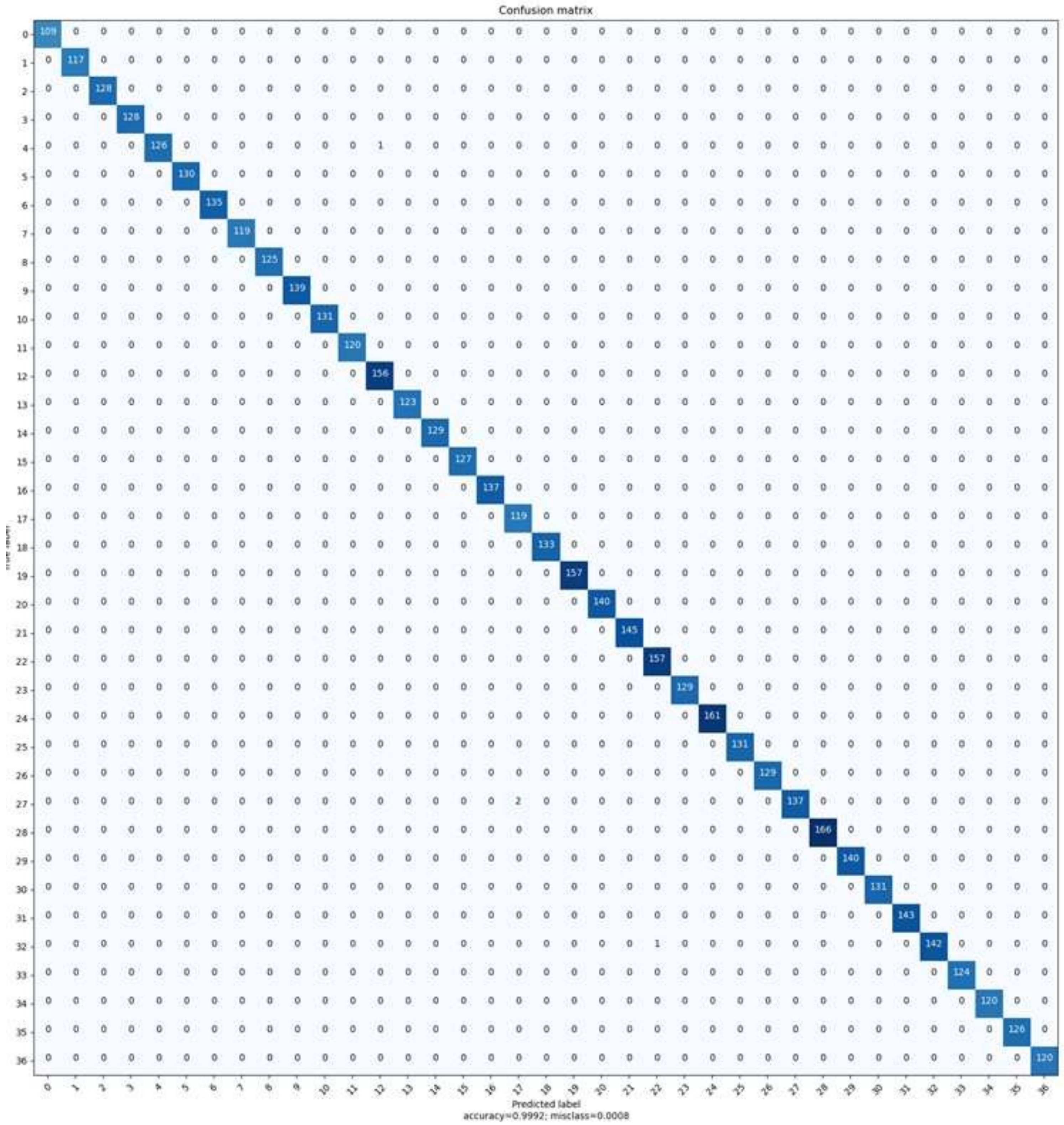
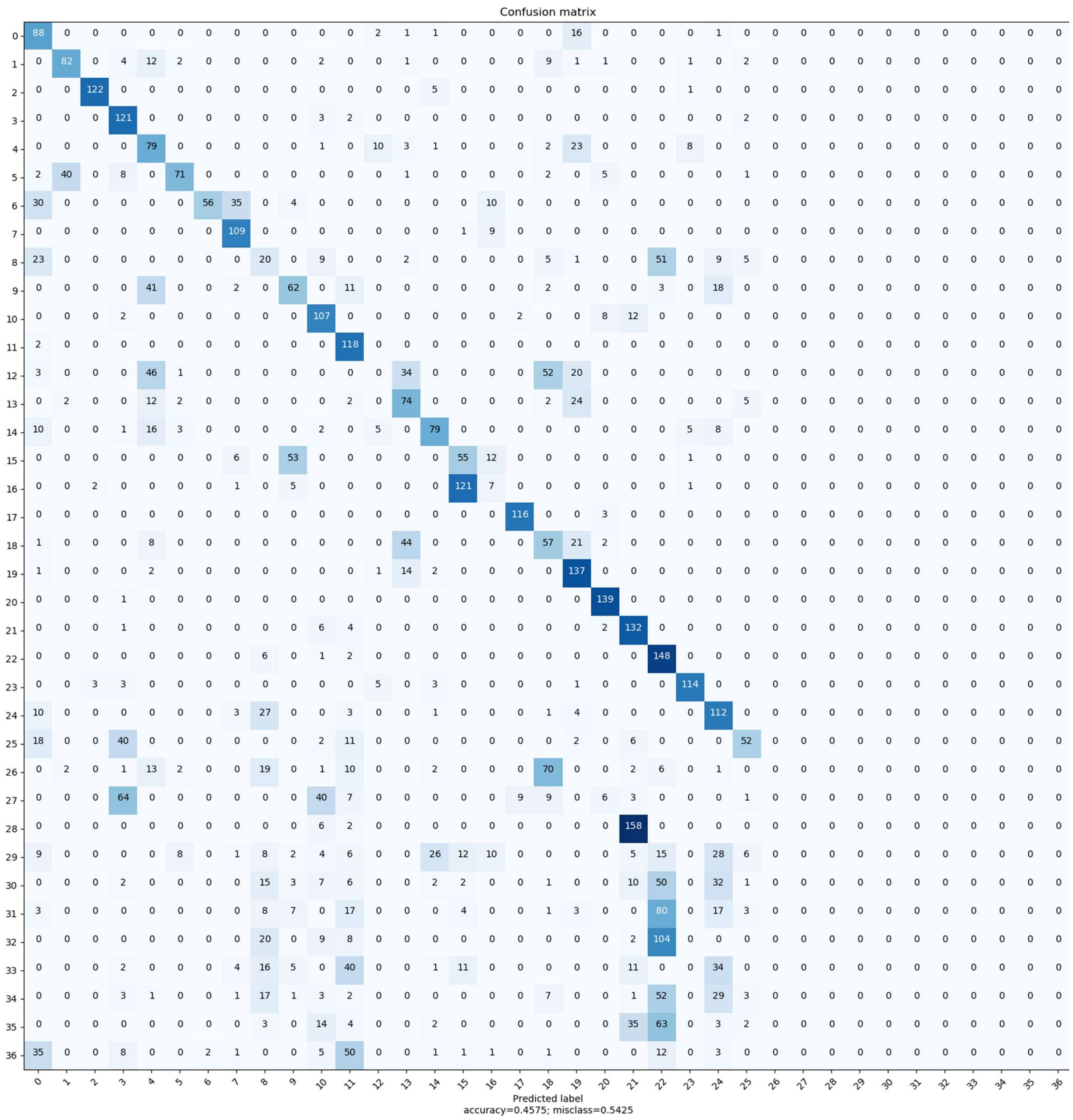


Fig 13. Confusion Matrix – validation accuracy of 99.92%(batch size 500)



Confusion matrix

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32	33	34	35	36
0	88	0	0	0	0	0	0	0	0	0	0	0	2	1	1	0	0	0	0	16	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0
1	0	82	0	4	12	2	0	0	0	0	2	0	0	1	0	0	0	0	9	1	1	0	0	1	0	2	0	0	0	0	0	0	0	0	0	0	0
2	0	0	122	0	0	0	0	0	0	0	0	0	0	0	5	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
3	0	0	0	121	0	0	0	0	0	0	3	2	0	0	0	0	0	0	0	0	0	0	0	0	0	2	0	0	0	0	0	0	0	0	0	0	0
4	0	0	0	0	79	0	0	0	0	0	1	0	10	3	1	0	0	0	2	23	0	0	0	8	0	0	0	0	0	0	0	0	0	0	0	0	0
5	2	40	0	8	0	71	0	0	0	0	0	0	0	1	0	0	0	0	2	0	5	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
6	30	0	0	0	0	0	56	35	0	4	0	0	0	0	0	0	10	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
7	0	0	0	0	0	0	0	109	0	0	0	0	0	0	0	1	9	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
8	23	0	0	0	0	0	0	0	20	0	9	0	0	2	0	0	0	0	5	1	0	0	51	0	9	5	0	0	0	0	0	0	0	0	0	0	0
9	0	0	0	0	41	0	0	2	0	62	0	11	0	0	0	0	0	0	2	0	0	0	3	0	18	0	0	0	0	0	0	0	0	0	0	0	0
10	0	0	0	2	0	0	0	0	0	0	107	0	0	0	0	0	0	2	0	0	8	12	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
11	2	0	0	0	0	0	0	0	0	0	0	118	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	3	0	0	0	46	1	0	0	0	0	0	0	0	34	0	0	0	0	52	20	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	0	2	0	0	12	2	0	0	0	0	0	2	0	74	0	0	0	0	2	24	0	0	0	0	0	5	0	0	0	0	0	0	0	0	0	0	0
14	10	0	0	1	16	3	0	0	0	0	2	0	5	0	79	0	0	0	0	0	0	0	0	5	8	0	0	0	0	0	0	0	0	0	0	0	0
15	0	0	0	0	0	0	0	6	0	53	0	0	0	0	0	55	12	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
16	0	0	2	0	0	0	0	1	0	5	0	0	0	0	0	121	7	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	116	0	0	3	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
18	1	0	0	0	8	0	0	0	0	0	0	0	0	44	0	0	0	0	57	21	2	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
19	1	0	0	0	2	0	0	0	0	0	0	0	1	14	2	0	0	0	0	137	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
20	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	139	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
21	0	0	0	1	0	0	0	0	0	0	6	4	0	0	0	0	0	0	0	0	2	132	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
22	0	0	0	0	0	0	0	0	6	0	1	2	0	0	0	0	0	0	0	0	0	0	148	0	0	0	0	0	0	0	0	0	0	0	0	0	0
23	0	0	3	3	0	0	0	0	0	0	0	0	5	0	3	0	0	0	0	1	0	0	0	114	0	0	0	0	0	0	0	0	0	0	0	0	0
24	10	0	0	0	0	0	0	3	27	0	0	3	0	0	1	0	0	0	1	4	0	0	0	0	112	0	0	0	0	0	0	0	0	0	0	0	0
25	18	0	0	40	0	0	0	0	0	0	2	11	0	0	0	0	0	0	0	2	0	6	0	0	0	52	0	0	0	0	0	0	0	0	0	0	0
26	0	2	0	1	13	2	0	0	19	0	1	10	0	0	2	0	0	0	70	0	0	2	6	0	1	0	0	0	0	0	0	0	0	0	0	0	0
27	0	0	0	64	0	0	0	0	0	0	40	7	0	0	0	0	0	9	9	0	6	3	0	0	0	1	0	0	0	0	0	0	0	0	0	0	0
28	0	0	0	0	0	0	0	0	0	0	6	2	0	0	0	0	0	0	0	0	0	158	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
29	9	0	0	0	0	8	0	1	8	2	4	6	0	0	26	12	10	0	0	0	0	5	15	0	28	6	0	0	0	0	0	0	0	0	0	0	0
30	0	0	0	2	0	0	0	0	15	3	7	6	0	0	2	2	0	0	1	0	0	10	50	0	32	1	0	0	0	0	0	0	0	0	0	0	0
31	3	0	0	0	0	0	0	0	8	7	0	17	0	0	0	4	0	0	1	3	0	0	80	0	17	3	0	0	0	0	0	0	0	0	0	0	0
32	0	0	0	0	0	0	0	0	20	0	9	8	0	0	0	0	0	0	0	0	0	2	104	0	0	0	0	0	0	0	0	0	0	0	0	0	0
33	0	0	0	2	0	0	0	4	16	5	0	40	0	0	1	11	0	0	0	0	0	11	0	0	34	0	0	0	0	0	0	0	0	0	0	0	0
34	0	0	0	3	1	0	0	1	17	1	3	2	0	0	0	0	0	0	7	0	0	1	52	0	29	3	0	0	0	0	0	0	0	0	0	0	0
35	0	0	0	0	0	0	0	0	3	0	14	4	0	0	2	0	0	0	0	0	0	35	63	0	3	2	0	0	0	0	0	0	0	0	0	0	0
36	35	0	0	8	0	0	2	1	0	0	5	50	0	0	1	1	1	0	1	0	0	0	12	0	3	0	0	0	0	0	0	0	0	0	0	0	0

Predicted label
accuracy=0.4575; misclass=0.5425

7. Conclusion and Future Scope

A successful Gesture Recognition system is identified by its accuracy and usability. Through this study, we present a fast, accurate, and efficient system capable of distinguishing between 37 gestures accurately. The model proposed uses simple OpenCV functions to replace the histograms, an unnecessary load over the whole system. Simple in-file functions (filtering, morphological closing, and thresholding) cut loose the need for an extra file for calculating histogram and make the whole mechanism much faster. Use of Convnet with just right parameters (batch size, no of epochs, etc.) gives us the high accuracy. Continuous, numerous testing with a taxing 37 gesture dataset testifies the 98% accuracy of our model. The confusion matrix shows the **validation accuracy of 99.92%** and a **misclass** of **just 0.08%.** Other performance metrics for the model are also satisfactory, with precision, recall, and f1-score all three being equal to 1.00. The same validation accuracy, on **reducing the batch size to 330**, **drops to 46%** and **misclassifies the gestures with 54%**.

The paper also highlights some milestone works previously done in the field. Exploring the methods using Histograms, it reveals their shortcomings and gives a better alternative. It tries new OpenCV functions apt for this task and shifts the focus back to these OpenCV functions.

For future endeavors in the same field, it is recommended to use a more sophisticated machine/pc/laptop having a **good GPU.** It can improve the performance of the system to a great extent by relieving the CPU of the graphics intensive work of image processing. GPUs, with their own dedicated memory, perform all the directed instructions. While the CPU tries to build deep learning models according to the user's instructions, the GPU can perform Graphics Rendering and all sorts of complex mathematical computations on the large Numpy arrays of images.

More studies like these will bring new advances in this area of research and one day we will be looking at a sophisticated application, which has integrated hand gesture recognition for Sign Languages.

Apart from helping the disabled, these recognition systems are also a great tool for HCI. We have seen some revolutionary tech in the HCI field recently. For e.g. the Motion Sensor in the latest Google Pixel 4. The phone has this amazing ability to monitor a user's gestures with its front camera and perform operations based on those gestures.

Highly sophisticated security systems can use gesture recognition systems for recognizing human gaits. Human Gaits are a combination of multiple gestures of different body parts, so we can supply Human Gaits to the model and recognize them in a similar fashion. Many movies have shown us the future of Gesture Recognition. The best being Tony Stark's gesture-controlled workplace. Such a future is awaiting us.

8. References

K. A. Bhaskaran, A. G. Nair, K. D. Ram, K. Ananthanarayanan and H. R. N. Vardhan, "Smart gloves for hand gesture recognition: Sign language to speech conversion system," 2016 International Conference on Robotics and Automation for Humanitarian Applications (RAHA), Kollam, 2016, pp. 1-6. doi 10.1109/RAHA.2016.7931887, URL: http://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=7931887&isnumber =7931858

Turkar, S., Peter, M., Kapakar, P., Dehankar, R., Sahu, P.H. (2017): Text Generation Using Hand Gesture Recognition" "International Journal of Informative & Futuristic Research (ISSN: 2347-1697), Vol. 4 No. (7), March 2017, pp. 6706-6711, Paper ID: IJIFR/V4/E7/013

K.-Y. Lian, C.-C. Chiu, Y.-J. Hong and W.-T. Sung, "Wearable armband for real time hand gesture recognition," in IEEE International Conference on Systems, Man, and Cybernetics, Canada, 2017.

Munasinghe, Nuwan. (2018). Dynamic Hand Gesture Recognition Using Computer Vision and Neural Networks. 10.1109/I2CT.2018.8529335.

Human Skin Detection Using RGB, HSV and YCbCr Color Models S. Kolkur1, D. Kalbande2, P.Shimpi2, C.Bapat2,and J. Jatakia 21 Department of Computer Engineering, Thadomal Shahani Engineering College, Bandra,Mumbai, India2Department of Computer Engineering, Sardar Patel Institute of Technology, Andheri,Mumbai, India {kolkur.seema@gmail.com;drkalbande@spit.ac.in; prajwalshimpi@gmail.com;chai.bapat@gmail.com;jatakiajanvi12@gmail.com}

Discrimination Between Skin and Non- Skin Pixels in Image Using the Range of HSV Color Space1Pooja Sharma, 2Dr. Veena Yadav1Dept. Of Computer Science, Jagannath University, Jaipur, Rajasthan, India2Dept. Of Computer Science, Global Institute of Engg. & Tech., Jaipur, Rajasthan, India

Saha, D.. (2018, May 9). Sign-Language (Version 1). figshare. https://doi.org/10.6084/m9.figshare.6241901.v1A very simple CNN project.

Manikandan, K. & Patidar, Ayush & Walia, Pallav & Barman Roy, Aneek. (2018). Hand Gesture Detection and Conversion to Speech and Text.

Hanwen Huang et al 2019 J. Phys.: Conf. Ser.1213 022001

International Journal of Applied Engineering Research ISSN 0973-4562 Volume 13, Number 9(2018) pp.7154-7161© Research India Publications. http://www.ripublication.com

Patel, P., & Patel, N. (2019). Vision Based Real-time Recognition of Hand Gestures for Indian Sign Language using Histogram of Oriented Gradients Features. International Journal of Next-Generation Computing, 10(2).

Hakim, N.L.; Shih, T.K.; Kasthuri Arachchi, S.P.; Aditya, W.; Chen, Y.-C.; Lin, C.-Y. Dynamic Hand Gesture Recognition Using 3DCNN and LSTM with FSM Context-Aware Model. Sensors 2019, 19, 5429.

C. Zhu, J. Yang, Z. Shao and C. Liu, "Vision based hand gesture recognition using 3d shape context," in IEEE/CAA Journal of Automatica Sinica, doi: 10.1109/JAS.2019.1911534.