CS512: Computer Vision

PROJECT

Virtual Calculator (Real-Time Hand Gesture Recognition Using

Finger Segmentation)

FNU SANDEEP

CWID: A20426557

TEJASBHAI PATEL

CWID: A20429780

❖ Abstract:

As we know that the transition world of this new era world is very much fast. So, new technology comes in, makes an impact and retires. Our Project named Virtual Calculator using Human Gestures is just an example of how the upcoming future will be that acts upon the gestures not the on the command that we give them manually. We just want to display an example of controlling the human-computer interaction with just simple hand gestures. So, we are in a path of replacing the mobile devices to a virtual device that will act as a mobile device and perform the same things what our mobile device does.

Isn't it would be great where we are on the path of developing a device that just not want commands from users, but it will read just his gestures and upon, that they are going to act?

❖ Problem Statement:

The Goal of developing such a kind of system is to showcase how we can make our devices smarter by just making them learn human Gestures. Here we are just showcasing a simple application of recognizing human gestures and performing some actions accordingly.

❖ Proposed Solution:

Gesture is a representation of physical behavior or emotional appearance. It includes body gesture and hand gesture. It falls into two categories: static gesture and dynamic gesture. For the former, the posture of the body or the gesture of the hand denotes a sign. For the latter, the movement of the body or the hand conveys some messages. Gesture can be used as a tool of communication between computer and human.

The Actual Methodology Involves:

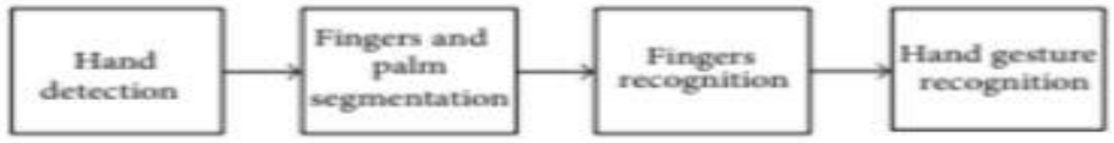


FIGURE 1: The overview of the proposed method for hand gesture recognition.

As shown in figure 1 the steps are going to be:

a) Hand Detection: In this method, we are going to differentiate between the background and the foreground elements. The skin color can be used to differentiate the hand region from the other moving objects. The color of the skin is measured with the HSV model. The HSV (hue, saturation, and value) value of the skin color is 315, 94, and 37, respectively.

b) Fingers and palm segmentation: The output of the hand detection is a binary image in which the white pixels are the members of the hand region, while the black pixels belong to the background. In the same step the following steps will be done.
    1) Palm Point
    2) Wrist Point
    3) Hand Rotation

c) Finger Recognition: This step will help us to know the number of Fingers used in the whole operation.

d) Hand Gestures Recognition: When the fingers are detected and recognized, the hand gesture can be recognized using a simple rule classifier. In the rule classifier, the hand gesture is predicted according to the number and content of fingers detected. The content of the fingers means what fingers are detected. The rule classifier is very effective and efficient. For example, if three fingers, that is, the middle finger, the ring finger, and the little finger, are detected, the hand gesture is classified as the label 3

e) Displaying the results: First, the number of Fingers will be displayed, and the output is shown on the same screen.

➢ **External Functions/ Library used:**

- **calcBackProject:**

The functions calcBackProject calculate the back project of the histogram. That is, similarly to calcHist, at each location (x, y) the function collects the values from the selected channels in the input images and finds the corresponding histogram bin. But instead of incrementing it, the function reads the bin value, scales it by scale, and stores in backProject(x,y) . In terms of statistics, the function computes probability of each element value in respect with the empirical probability distribution represented by the histogram. See how, for example, you can find and track a bright-colored object in a scene:

1. Before tracking, show the object to the camera so that it covers almost the whole frame. Calculate a hue histogram. The histogram may have strong maximums, corresponding to the dominant colors in the object.

2. When tracking, calculate a back projection of a hue plane of each input video frame using that pre-computed histogram. Threshold the back projection to suppress weak colors. It may also make sense to suppress pixels with non-sufficient color saturation and too dark or too bright pixels.

3. Find connected components in the resulting picture and choose, for example, the largest component.

This is an approximate algorithm of the CamShift() color object tracker.

- **getStructuringElement:**

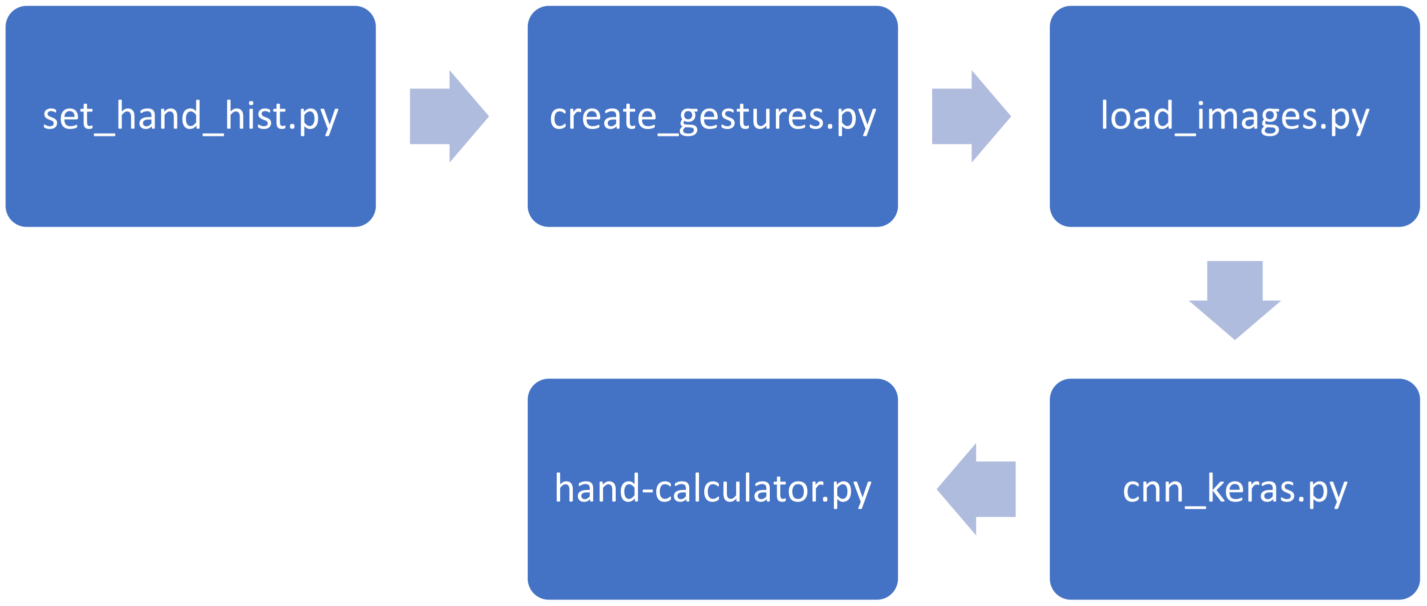
The function constructs and returns the structuring element that can be further passed to erode, dilate or morphologyEx. But you can also construct an arbitrary binary mask yourself and use it as the structuring element.

- **Pickle:**

The pickle module implements a fundamental, but powerful algorithm for serializing and de-serializing a Python object structure.

1. **Pickling -** is the process whereby a Python object hierarchy is converted into a byte stream, and **Unpickling -** is the inverse operation, whereby a byte stream is converted back into an object hierarchy.
2. Pickling (and unpickling) is alternatively known as **serialization**, **marshalling**, or **flattening**.

❖ Implementation Details:



---

1. Description of Set_hand_hist.py:

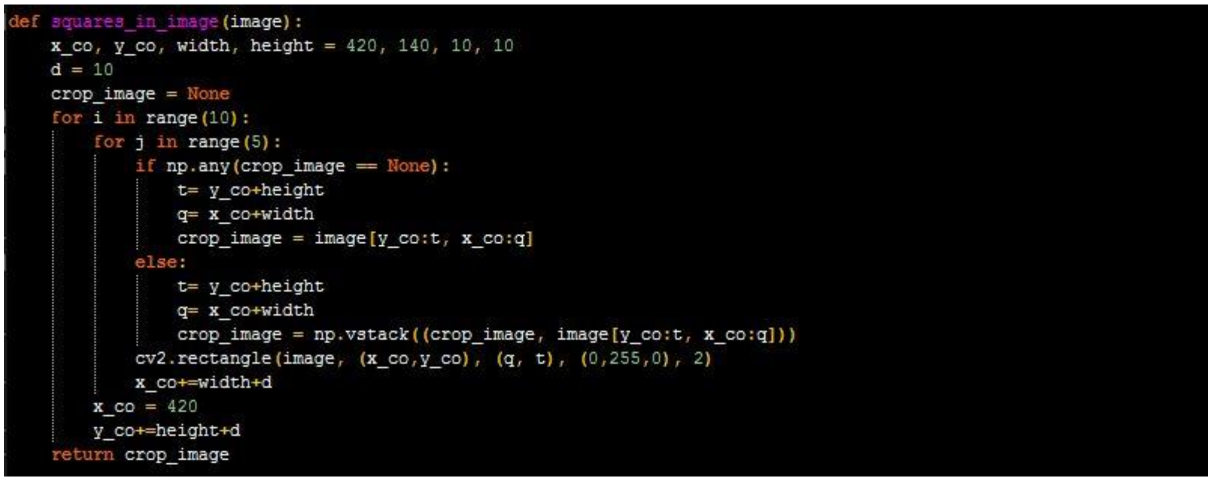
This code file consists of two parts: -

a) The part on the screen from where the threshold of the Image needs to be computed

b) This is the part which computes the histogram value and save that value to file name hist.

Code Explanation with algorithm:

1)Firstly, we will compute the threshold of the image. But, there is some constraint in this part that the threshold value for the whole screen will look very different than the threshold value of certain part. So, we opt for the certain Area Threshold value computation.



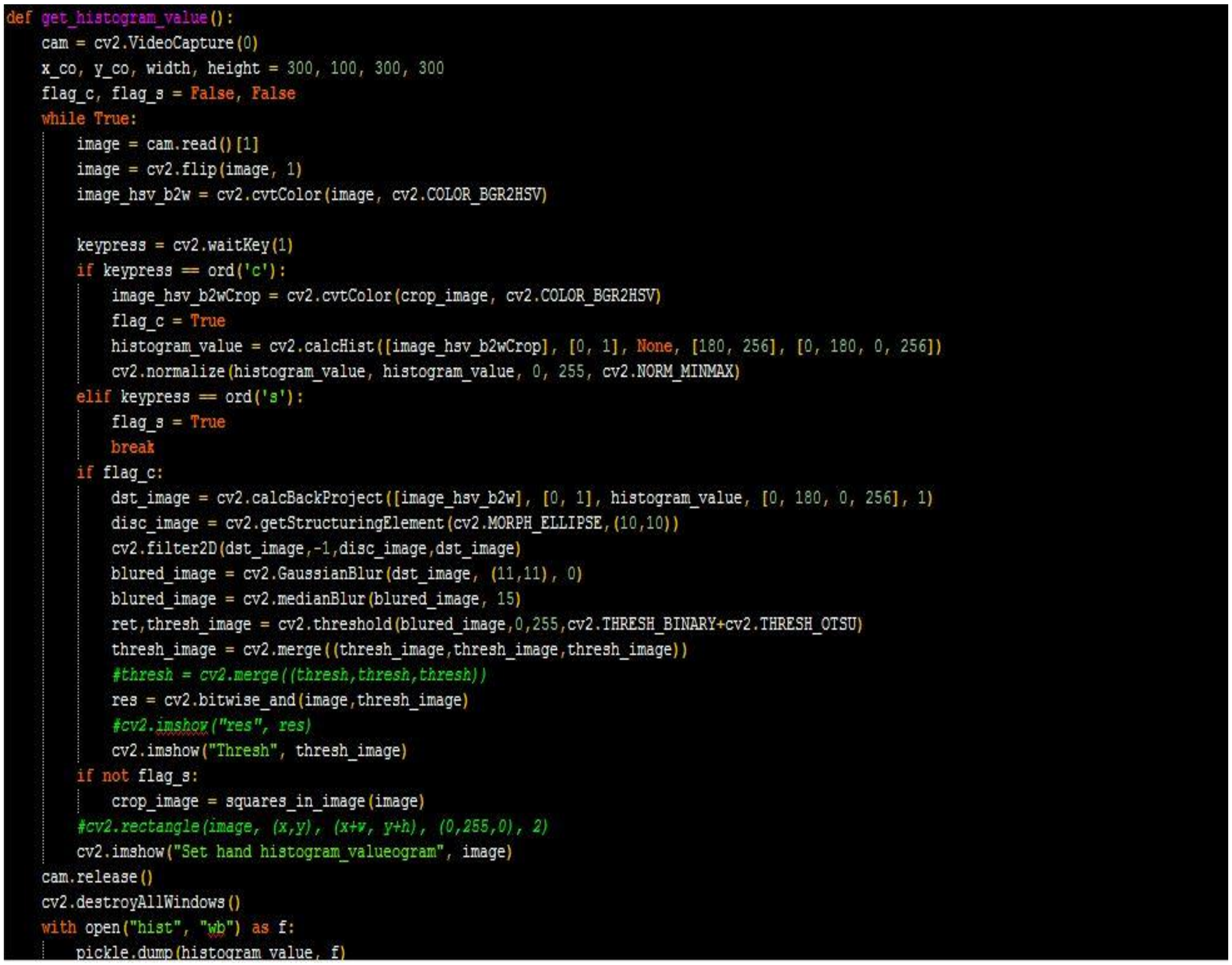
```python
def squares_in_image(image):
    x_co, y_co, width, height = 420, 140, 10, 10
    d = 10
    crop_image = None
    for i in range(10):
        for j in range(5):
            if np.any(crop_image == None):
                t= y_co+height
                q= x_co+width
                crop_image = image[y_co:t, x_co:q]
            else:
                t= y_co+height
                q= x_co+width
                crop_image = np.vstack((crop_image, image[y_co:t, x_co:q]))
            cv2.rectangle(image, (x_co,y_co), (q, t), (0,255,0), 2)
            x_co+=width+d
        x_co = 420
        y_co+=height+d
    return crop_image
```

Here, we defined the Area which will compute the threshold value. which will be helpful in processing all computation after this.

This code simply says that Image that exist between x_co, y_co, width, height will take into account for the computation. The area is indicated by the small squares on the output screen.

2) Second Part of this file computes the histogram value. Here the threshold values are computed on the live video display.

Code Explanation with Algorithm



```
def get_histogram_value():
    cam = cv2.VideoCapture(0)
    x_co, y_co, width, height = 300, 100, 300, 300
    flag_c, flag_s = False, False
    while True:
        image = cam.read()[1]
        image = cv2.flip(image, 1)
        image_hsv_b2w = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

        keypress = cv2.waitKey(1)
        if keypress == ord('c'):
            image_hsv_b2wCrop = cv2.cvtColor(crop_image, cv2.COLOR_BGR2HSV)
            flag_c = True
            histogram_value = cv2.calcHist([image_hsv_b2wCrop], [0, 1], None, [180, 256], [0, 180, 0, 256])
            cv2.normalize(histogram_value, histogram_value, 0, 255, cv2.NORM_MINMAX)
        elif keypress == ord('s'):
            flag_s = True
            break
        if flag_c:
            dst_image = cv2.calcBackProject([image_hsv_b2w], [0, 1], histogram_value, [0, 180, 0, 256], 1)
            disc_image = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10,10))
            cv2.filter2D(dst_image,-1,disc_image,dst_image)
            blured_image = cv2.GaussianBlur(dst_image, (11,11), 0)
            blured_image = cv2.medianBlur(blured_image, 15)
            ret,thresh_image = cv2.threshold(blured_image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)
            thresh_image = cv2.merge((thresh_image,thresh_image,thresh_image))
            #thresh = cv2.merge((thresh,thresh,thresh))
            res = cv2.bitwise_and(image,thresh_image)
            #cv2.imshow("res", res)
            cv2.imshow("Thresh", thresh_image)
        if not flag_s:
            crop_image = squares_in_image(image)
        #cv2.rectangle(image, (x,y), (x+w, y+h), (0,255,0), 2)
        cv2.imshow("Set hand histogram_valueogram", image)
    cam.release()
    cv2.destroyAllWindows()
    with open("hist", "wb") as f:
        pickle.dump(histogram_value, f)
```

Here the program starts with video capturing

We will compute each frame of video and at the same time we will compute threshold value of the same frame.

The computation will start by pressing “c” and then saved by pressing “s”

Algorithm:

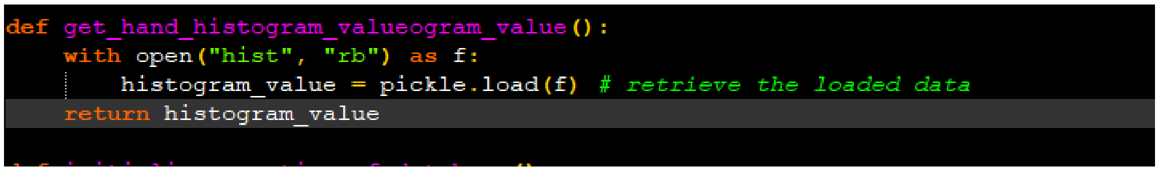
1) convert the image frame into Gray Scale.

2) If “c” is pressed, then:
    (i) Hist value will be computed for that frame and normalize it with the output
    (ii) Blurring of the image is done i.e. filters are applied.
    (iii) Threshold values are computed for each frame.

3) If “s” is pressed then the state is saved to the file named hist and it will terminate the program

2. Create Gestures:

- To create your gestures run the command “python create_gestures.py”.



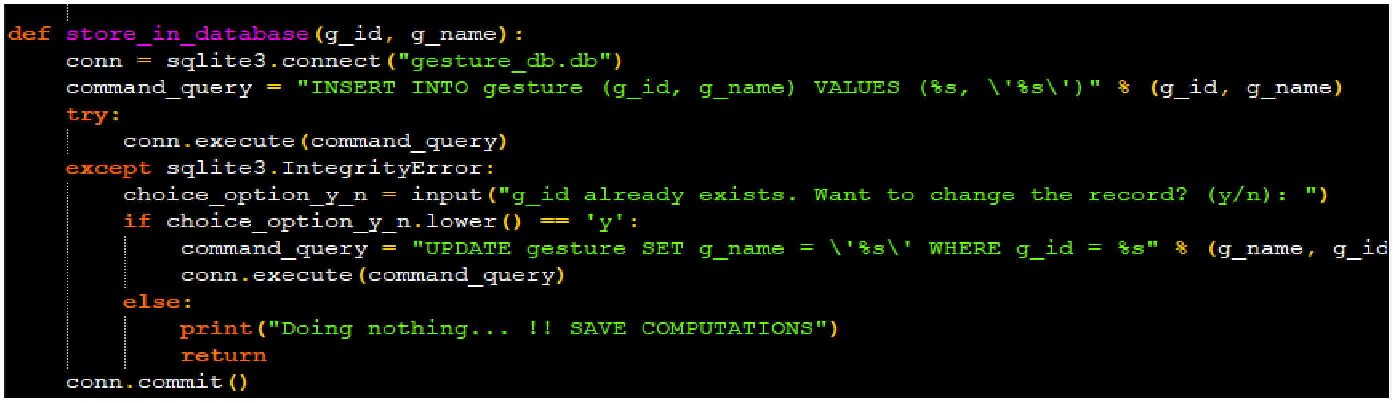
```python
def get_hand_histogram_valueogram_value():
    with open("hist", "rb") as f:
        histogram_value = pickle.load(f) # retrieve the loaded data
    return histogram_value
```

It will retrieve the histogram values.

- It will create a database in the home directory under the name gestures.

- After the execution of this program, you will have to enter the gesture number and gesture name/text.

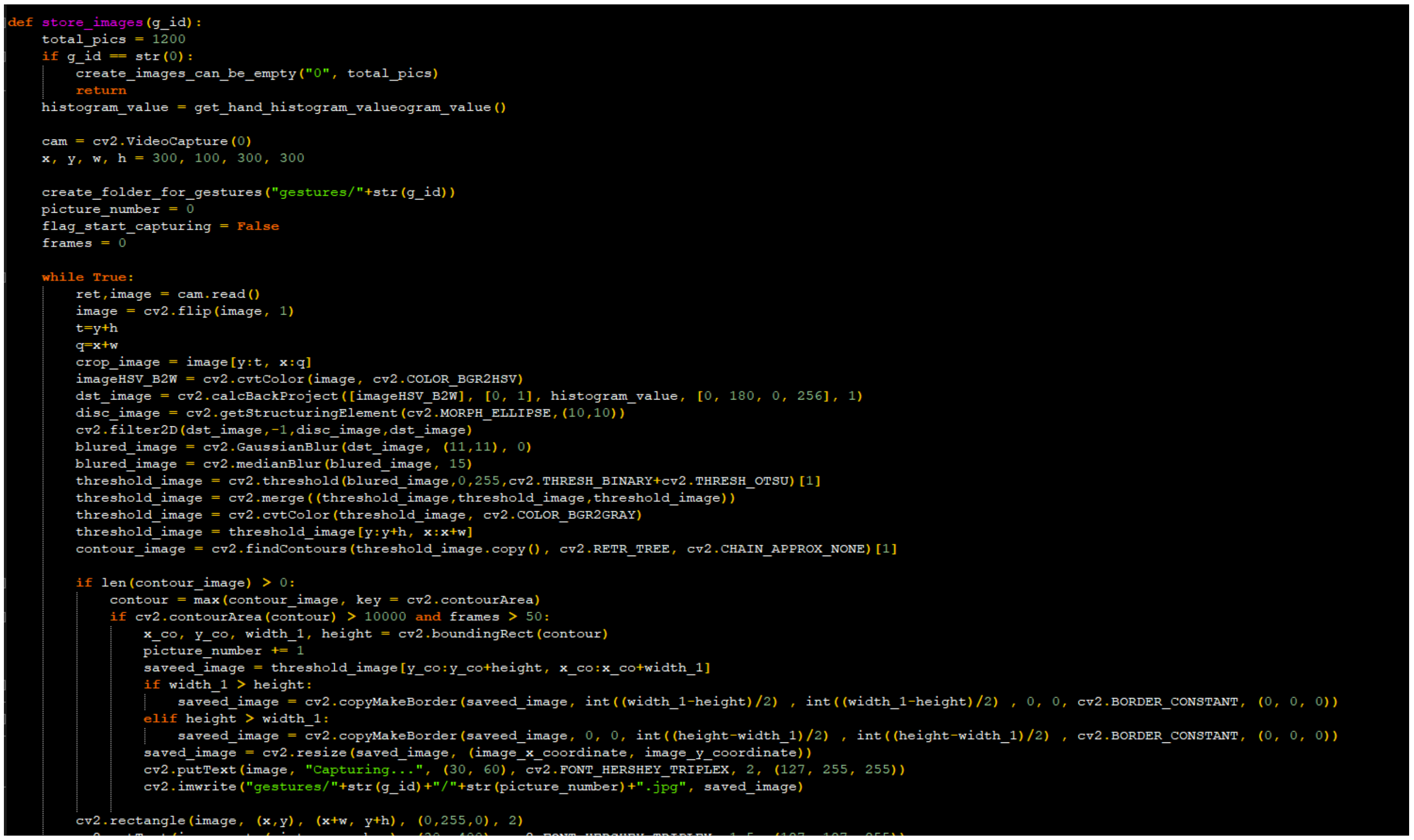
- It will create the sub folder under the name “gesture number” that you provided.



```python
def store_in_database(g_id, g_name):
    conn = sqlite3.connect("gesture_db.db")
    command_query = "INSERT INTO gesture (g_id, g_name) VALUES (%s, \'%s\')" % (g_id, g_name)
    try:
        conn.execute(command_query)
    except sqlite3.IntegrityError:
        choice_option_y_n = input("g_id already exists. Want to change the record? (y/n): ")
        if choice_option_y_n.lower() == 'y':
            command_query = "UPDATE gesture SET g_name = \'%s\' WHERE g_id = %s" % (g_name, g_id
            conn.execute(command_query)
        else:
            print("Doing nothing... !! SAVE COMPUTATIONS")
            return
    conn.commit()
```

- Then an OpenCV window called "Capturing gestures" which will appear. In the webcam feed you will see a green window (inside which you will have to do your gesture) and a counter that counts the number of pictures stored.

- Press 'c' when you are ready with your gesture. Capturing gesture will begin after a few seconds. Move your hand a little bit here and there. You can pause capturing by pressing 'c' and resume it by pressing 'c'.



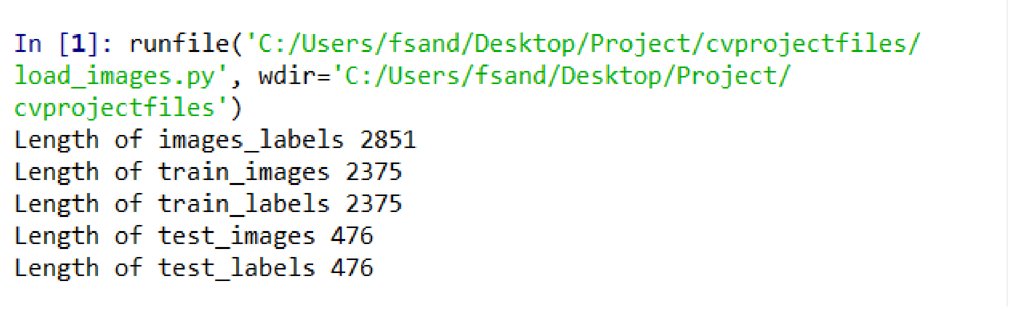
```
def store_images(g_id):
    total_pics = 1200
    if g_id == str(0):
        create_images_can_be_empty("0", total_pics)
        return
    histogram_value = get_hand_histogram_valueogram_value()

    cam = cv2.VideoCapture(0)
    x, y, w, h = 300, 100, 300, 300

    create_folder_for_gestures("gestures/"+str(g_id))
    picture_number = 0
    flag_start_capturing = False
    frames = 0

    while True:
        ret,image = cam.read()
        image = cv2.flip(image, 1)
        t=y+h
        q=x+w
        crop_image = image[y:t, x:q]
        imageHSV_B2W = cv2.cvtColor(image, cv2.COLOR_BGR2HSV)
        dst_image = cv2.calcBackProject([imageHSV_B2W], [0, 1], histogram_value, [0, 180, 0, 256], 1)
        disc_image = cv2.getStructuringElement(cv2.MORPH_ELLIPSE,(10,10))
        cv2.filter2D(dst_image,-1,disc_image,dst_image)
        blured_image = cv2.GaussianBlur(dst_image, (11,11), 0)
        blured_image = cv2.medianBlur(blured_image, 15)
        threshold_image = cv2.threshold(blured_image,0,255,cv2.THRESH_BINARY+cv2.THRESH_OTSU)[1]
        threshold_image = cv2.merge((threshold_image,threshold_image,threshold_image))
        threshold_image = cv2.cvtColor(threshold_image, cv2.COLOR_BGR2GRAY)
        threshold_image = threshold_image[y:y+h, x:x+w]
        contour_image = cv2.findContours(threshold_image.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)[1]

        if len(contour_image) > 0:
            contour = max(contour_image, key = cv2.contourArea)
            if cv2.contourArea(contour) > 10000 and frames > 50:
                x_co, y_co, width_1, height = cv2.boundingRect(contour)
                picture_number += 1
                saveed_image = threshold_image[y_co:y_co+height, x_co:x_co+width_1]
                if width_1 > height:
                    saveed_image = cv2.copyMakeBorder(saveed_image, int((width_1-height)/2) , int((width_1-height)/2) , 0, 0, cv2.BORDER_CONSTANT, (0, 0, 0))
                elif height > width_1:
                    saveed_image = cv2.copyMakeBorder(saveed_image, 0, 0, int((height-width_1)/2) , int((height-width_1)/2) , cv2.BORDER_CONSTANT, (0, 0, 0))
                saved_image = cv2.resize(saved_image, (image_x_coordinate, image_y_coordinate))
                cv2.putText(image, "Capturing...", (30, 60), cv2.FONT_HERSHEY_TRIPLEX, 2, (127, 255, 255))
                cv2.imwrite("gestures/"+str(g_id)+"/"+str(picture_number)+".jpg", saved_image)

        cv2.rectangle(image, (x,y), (x+w, y+h), (0,255,0), 2)
```

- Capturing resumes after a few second, after the counter reaches 1200 the window will close automatically.

- All the images will then store in our database.

- When you are done adding new gestures run the load_images.py file once. You do not need to run this file again until and unless you add a new gesture.

3. Description of load_images.py:

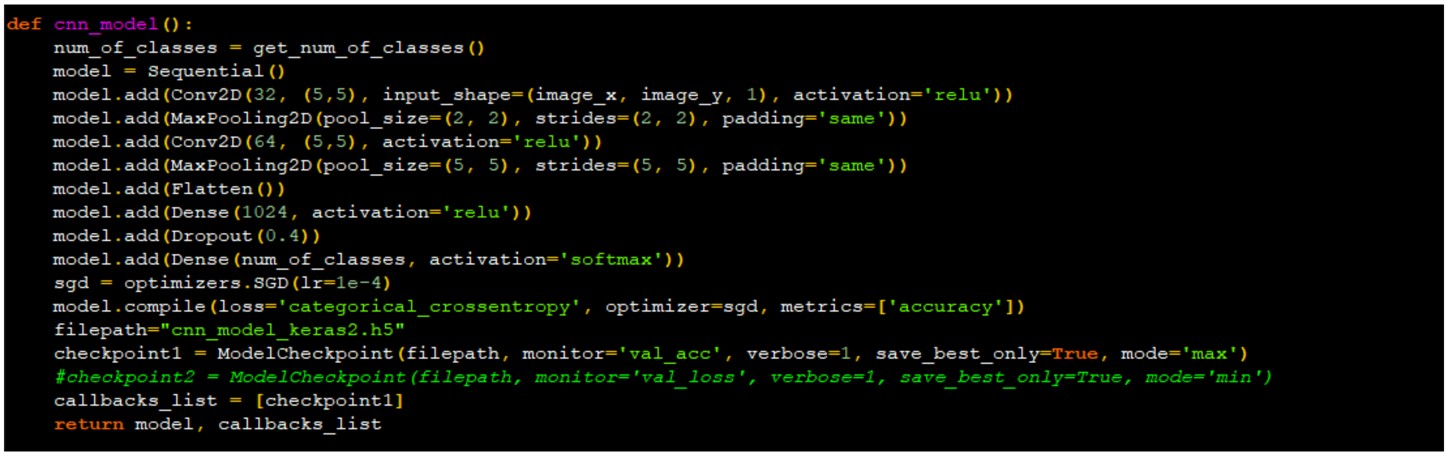


```
In [1]: runfile('C:/Users/fsand/Desktop/Project/cvprojectfiles/
load_images.py', wdir='C:/Users/fsand/Desktop/Project/
cvprojectfiles')
Length of images_labels 2851
Length of train_images 2375
Length of train_labels 2375
Length of test_images 476
Length of test_labels 476
```

It will give the number of all the images stored in the database.

4. Description for cnn_keras.py:

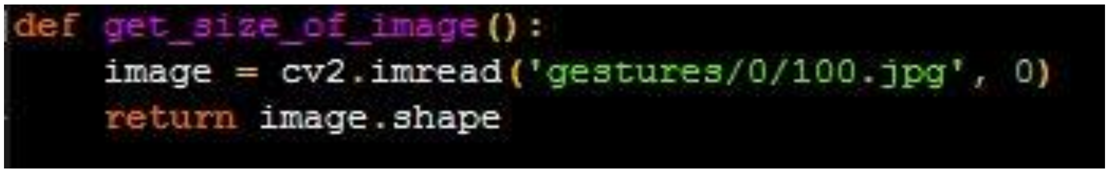
- In this we are training the model using 2 layers of filters and pooling each.
- Dropout rate of 40%
- Gradient descent optimization and a learning rate of (0.0001)



```python
def cnn_model():
    num_of_classes = get_num_of_classes()
    model = Sequential()
    model.add(Conv2D(32, (5,5), input_shape=(image_x, image_y, 1), activation='relu'))
    model.add(MaxPooling2D(pool_size=(2, 2), strides=(2, 2), padding='same'))
    model.add(Conv2D(64, (5,5), activation='relu'))
    model.add(MaxPooling2D(pool_size=(5, 5), strides=(5, 5), padding='same'))
    model.add(Flatten())
    model.add(Dense(1024, activation='relu'))
    model.add(Dropout(0.4))
    model.add(Dense(num_of_classes, activation='softmax'))
    sgd = optimizers.SGD(lr=1e-4)
    model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
    filepath="cnn_model_keras2.h5"
    checkpoint1 = ModelCheckpoint(filepath, monitor='val_acc', verbose=1, save_best_only=True, mode='max')
    #checkpoint2 = ModelCheckpoint(filepath, monitor='val_loss', verbose=1, save_best_only=True, mode='min')
    callbacks_list = [checkpoint1]
    return model, callbacks_list
```

5. Description for hand-calculator.py:

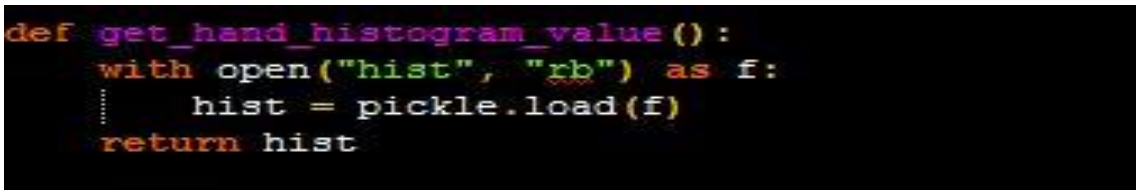
1)



```python
def get_size_of_image():
    image = cv2.imread('gestures/0/100.jpg', 0)
    return image.shape
```

This is the first module where we get the size of the image.

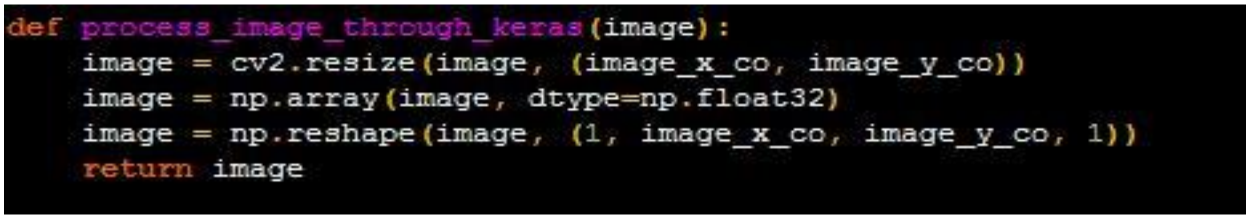
2)



```python
def get_hand_histogram_value():
    with open("hist", "rb") as f:
        hist = pickle.load(f)
    return hist
```

This function is to get the histogram value that is set in the file named “hist”. This file is very important for running this program. Because if this file is not there or if there is any kind of problem in reading this file then we will not be able to get the threshold value which are set for particular gestures.

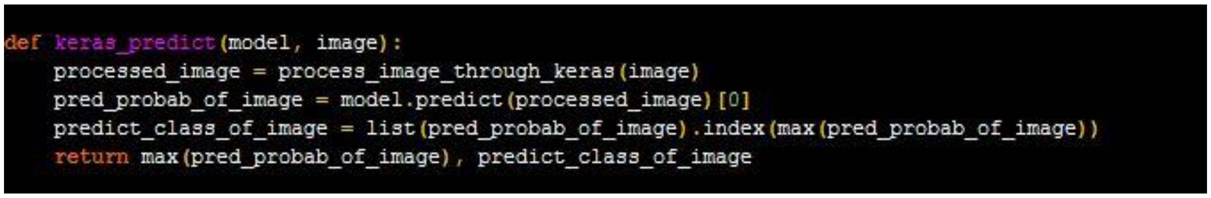
3)



```
def process_image_through_keras(image):
    image = cv2.resize(image, (image_x_co, image_y_co))
    image = np.array(image, dtype=np.float32)
    image = np.reshape(image, (1, image_x_co, image_y_co, 1))
    return image
```

This function is for resizing and reshaping the image. This function will convert the image to size that is available to train model. If the size of the image is not proper then though we get the same histogram for the image but, still we will not get the proper output.

4)

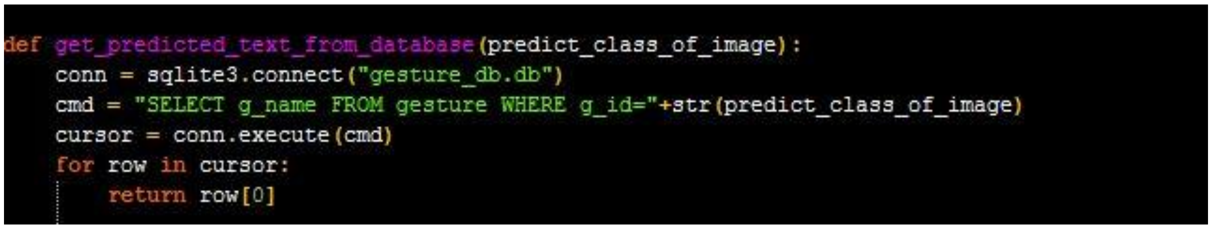


```
def keras_predict(model, image):
    processed_image = process_image_through_keras(image)
    pred_probab_of_image = model.predict(processed_image)[0]
    predict_class_of_image = list(pred_probab_of_image).index(max(pred_probab_of_image))
    return max(pred_probab_of_image), predict_class_of_image
```

This function a core function of this project as it uses the model that is trained on set of images and after that it predicts two things:

1) Predict that it belongs to particular image (probability that particular image is good)
2) Predict that this particular image belongs to that particular class labels

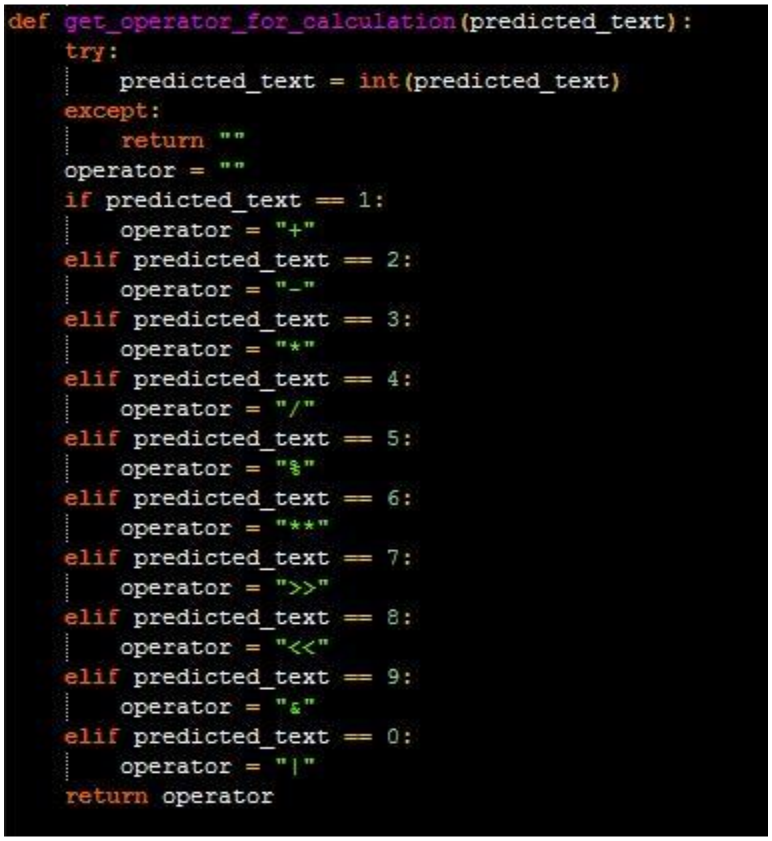
5)



```
def get_predicted_text_from_database(predict_class_of_image):
    conn = sqlite3.connect("gesture_db.db")
    cmd = "SELECT g_name FROM gesture WHERE g_id="+str(predict_class_of_image)
    cursor = conn.execute(cmd)
    for row in cursor:
        return row[0]
```

This function performs the task of predicting the text from the database. This function connects to the gestures database and through that database we get particular class of image to which that image belongs to.

6)



```python
def get_operator_for_calculation(predicted_text):
    try:
        predicted_text = int(predicted_text)
    except:
        return ""
    operator = ""
    if predicted_text == 1:
        operator = "+"
    elif predicted_text == 2:
        operator = "-"
    elif predicted_text == 3:
        operator = "*"
    elif predicted_text == 4:
        operator = "/"
    elif predicted_text == 5:
        operator = "%"
    elif predicted_text == 6:
        operator = "**"
    elif predicted_text == 7:
        operator = ">>"
    elif predicted_text == 8:
        operator = "<<"
    elif predicted_text == 9:
        operator = "&"
    elif predicted_text == 0:
        operator = "|"
    return operator
```

This function deals with the prediction of the operator from the predicted class. During operator selection, 1 means '+', 2 means '-', 3 means '*', 4 means '/', 5 means '%', 6 means '**', 7 means '>>' or right shift operator, 8 means '<<' or left shift operator, 9 means '&' or bitwise AND and 0 means '|' or bitwise OR.

7)

For the main Application of this project which do the task of user-interface and also does the task of displaying the output.

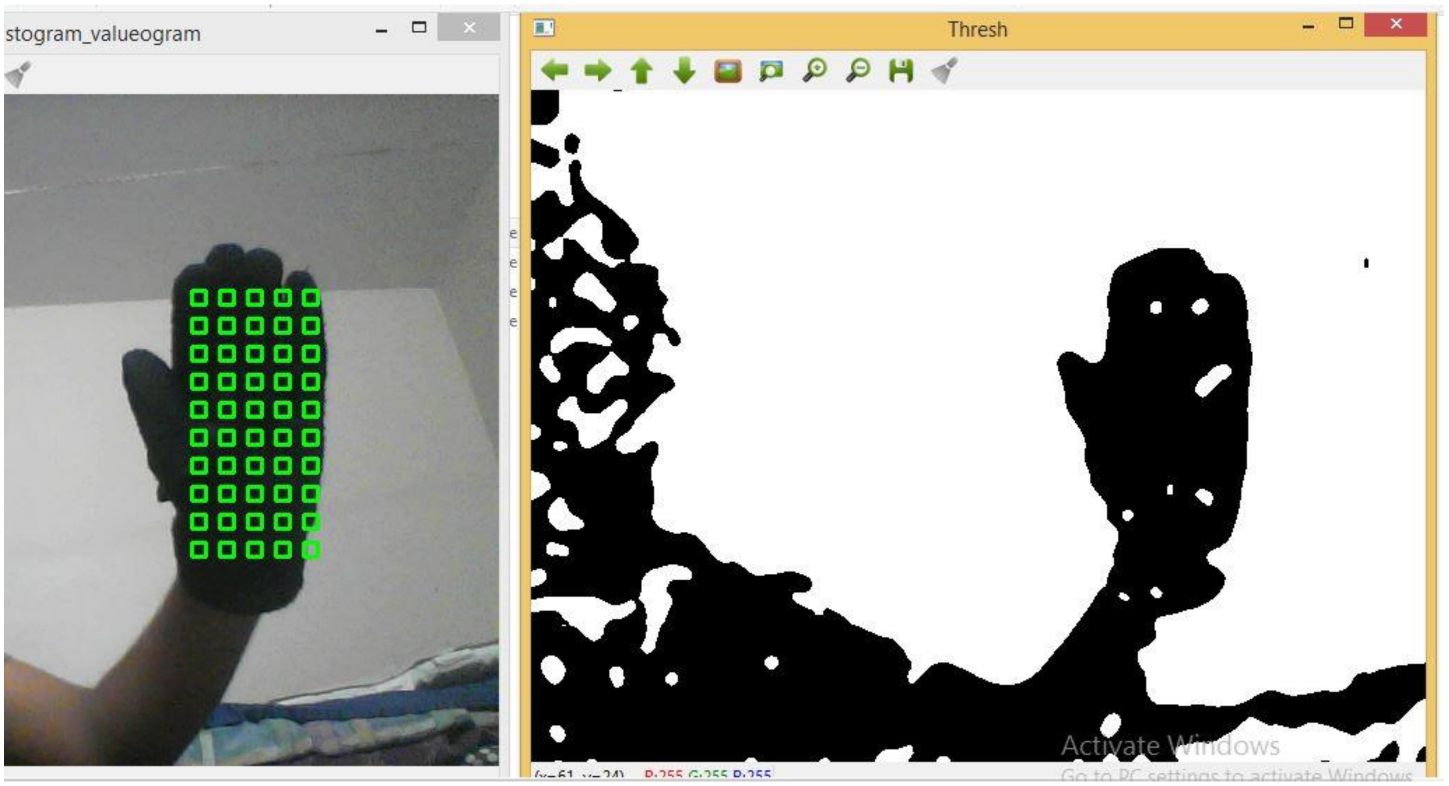
The algorithm for that is given bellow:

1) First video is captured for the processing and divided into frames after that it is converted to the gray scale then blurring is applied and after that the threshold of the image is computed.
2) From the threshold image contours of the image is computed.
   If contours > 10000
    1) then predicted probability will be checked and if it is greater than 70% then we get the predicted label
    2) if predicted text is confirmed then enter that number on the screen and also if the same frame remains constant then that number will be confirmed otherwise not.
    3) For operator confirmation the same rule will be applied as previous one. But, the only difference is that this time operator will be opted from the operator list.
    4) Again, for the second number we will be performing the same action as the step number 2

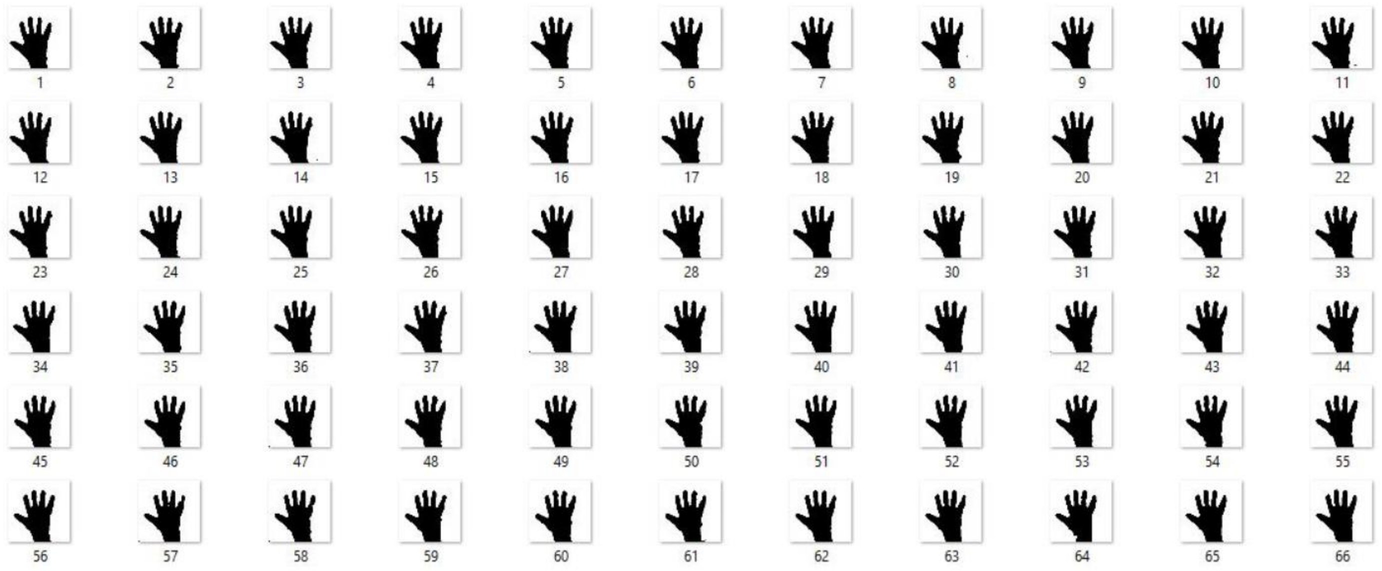
3) After getting the calculation done if the state remains same for 30 frames then automatically it will start processing new numbers.

❖ **Results:**

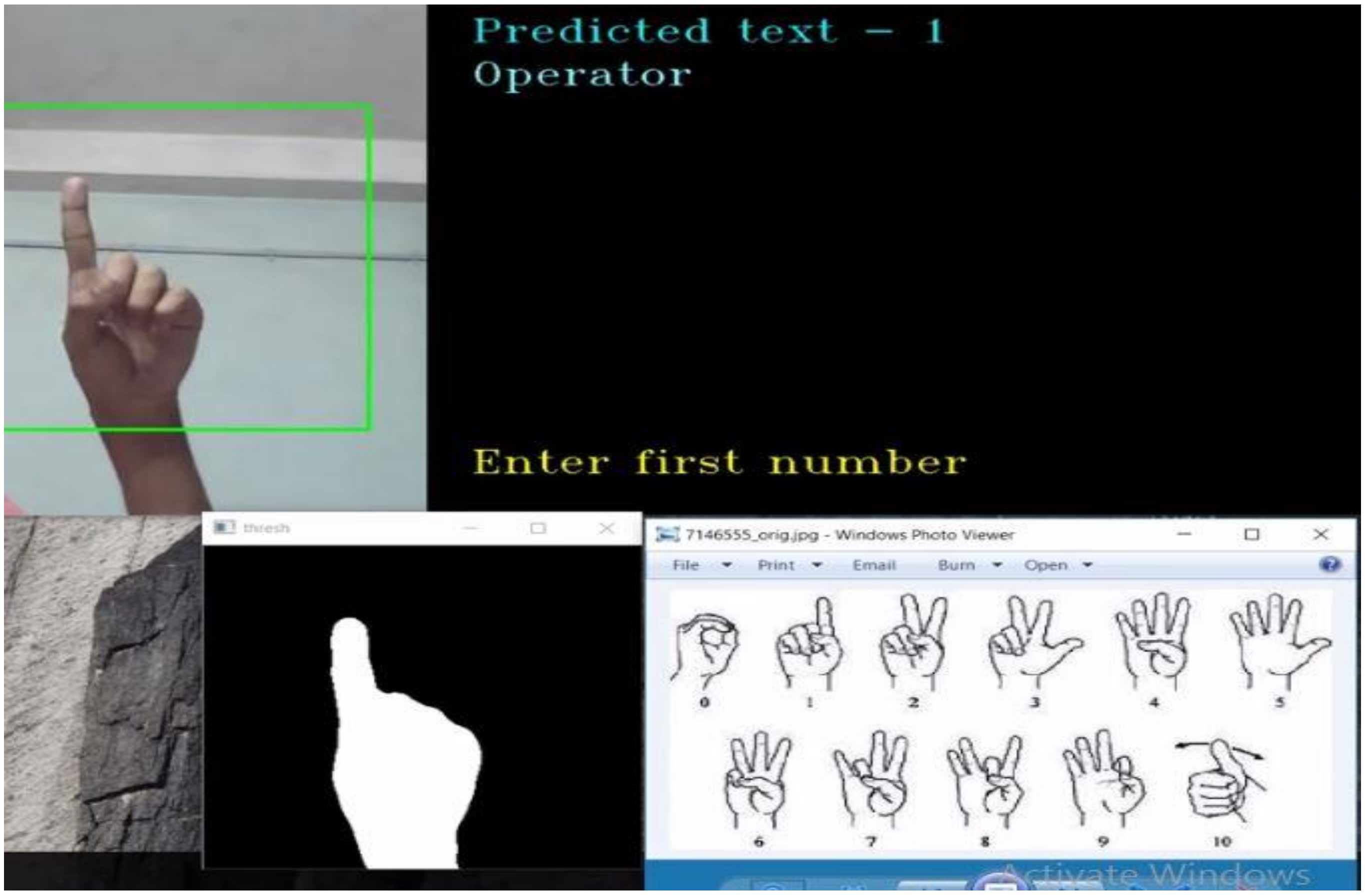
1. Histogram



2. Gestures stored in database



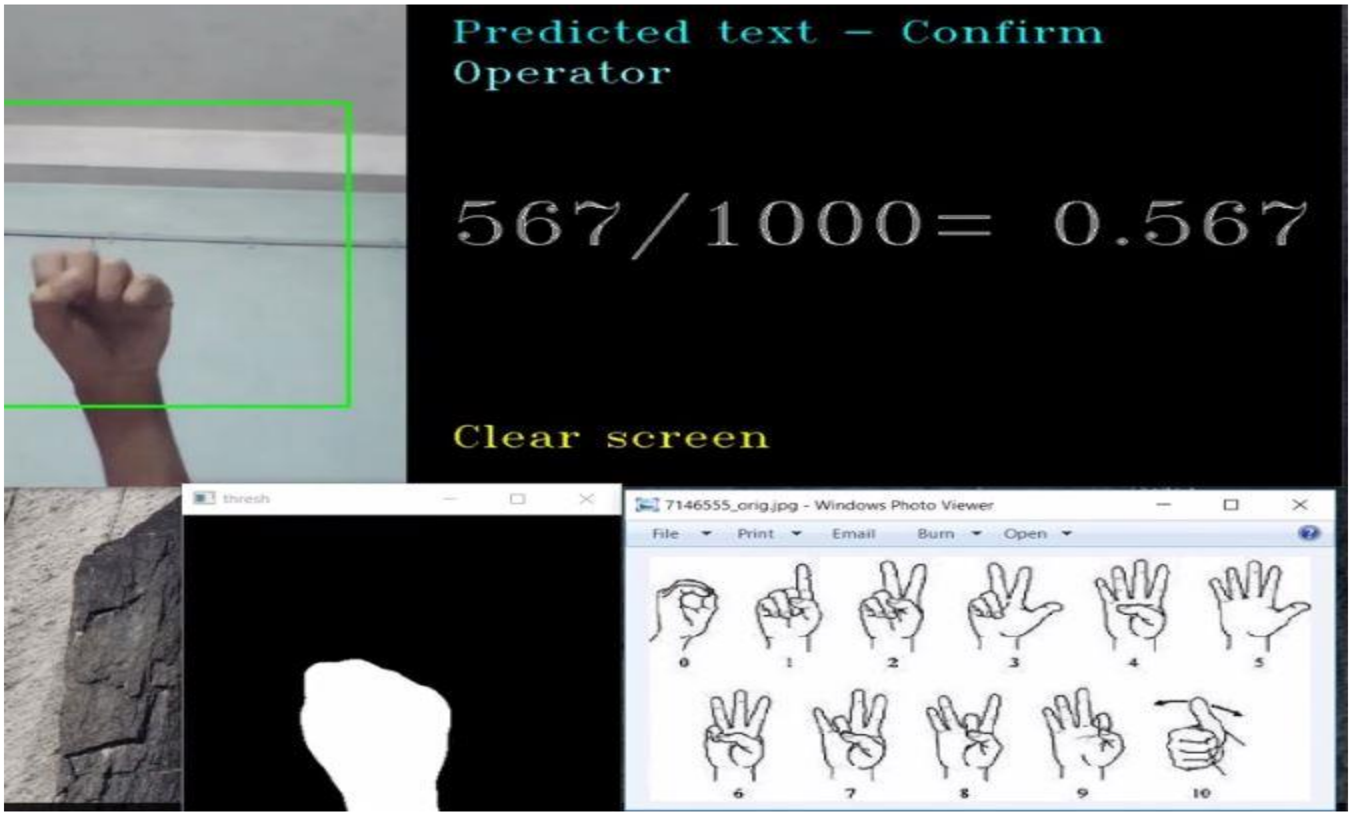
3. Predicting value



```
Predicted text - 1
Operator

Enter first number
```

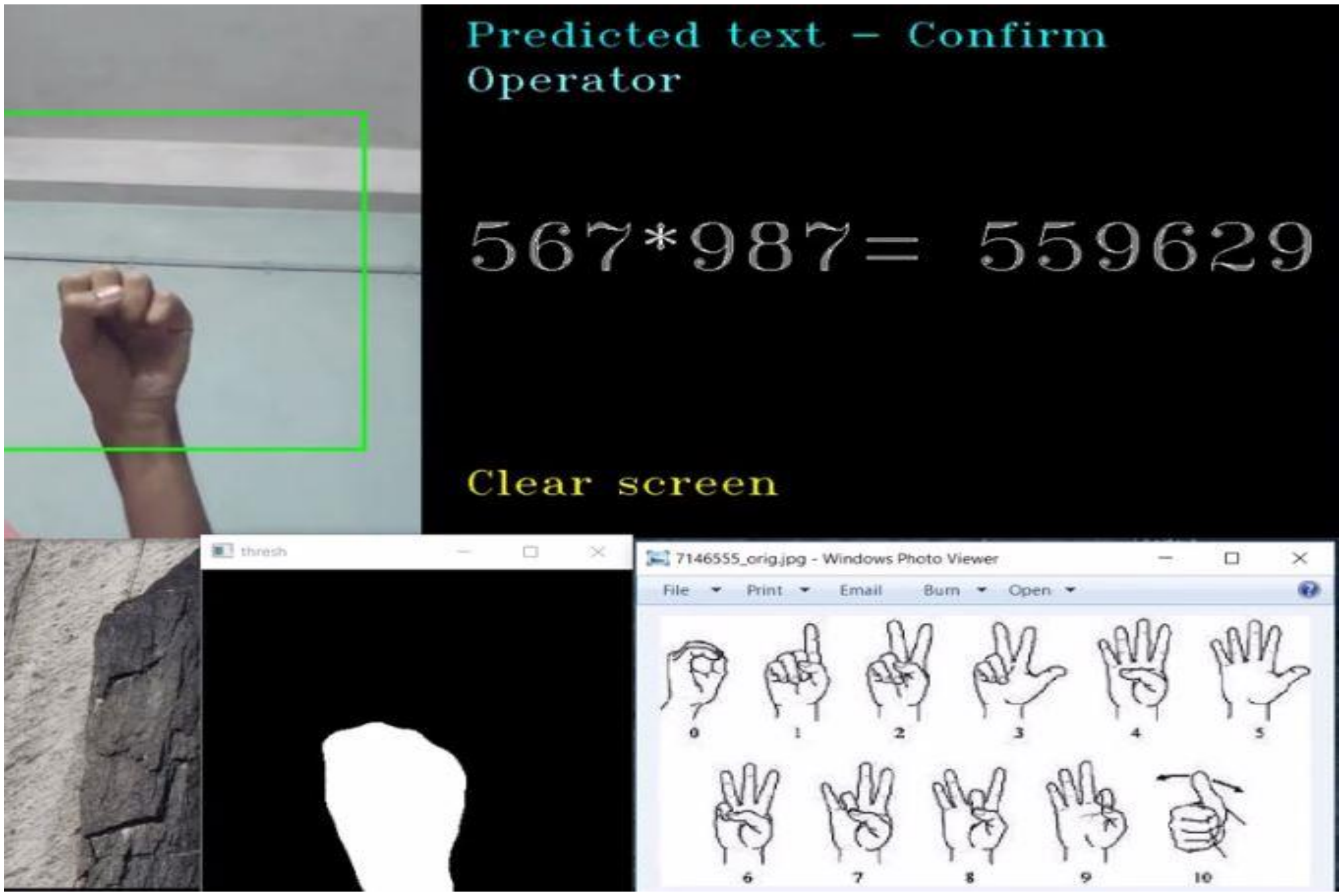
thresh
7146555_orig.jpg - Windows Photo Viewer
File Print Email Burn Open
0
1
2
3
4
5
6
7
8
9
10
Activate Windows

4. Hand Calculator:



```
Predicted text - Confirm
Operator

567/1000= 0.567

Clear screen
```

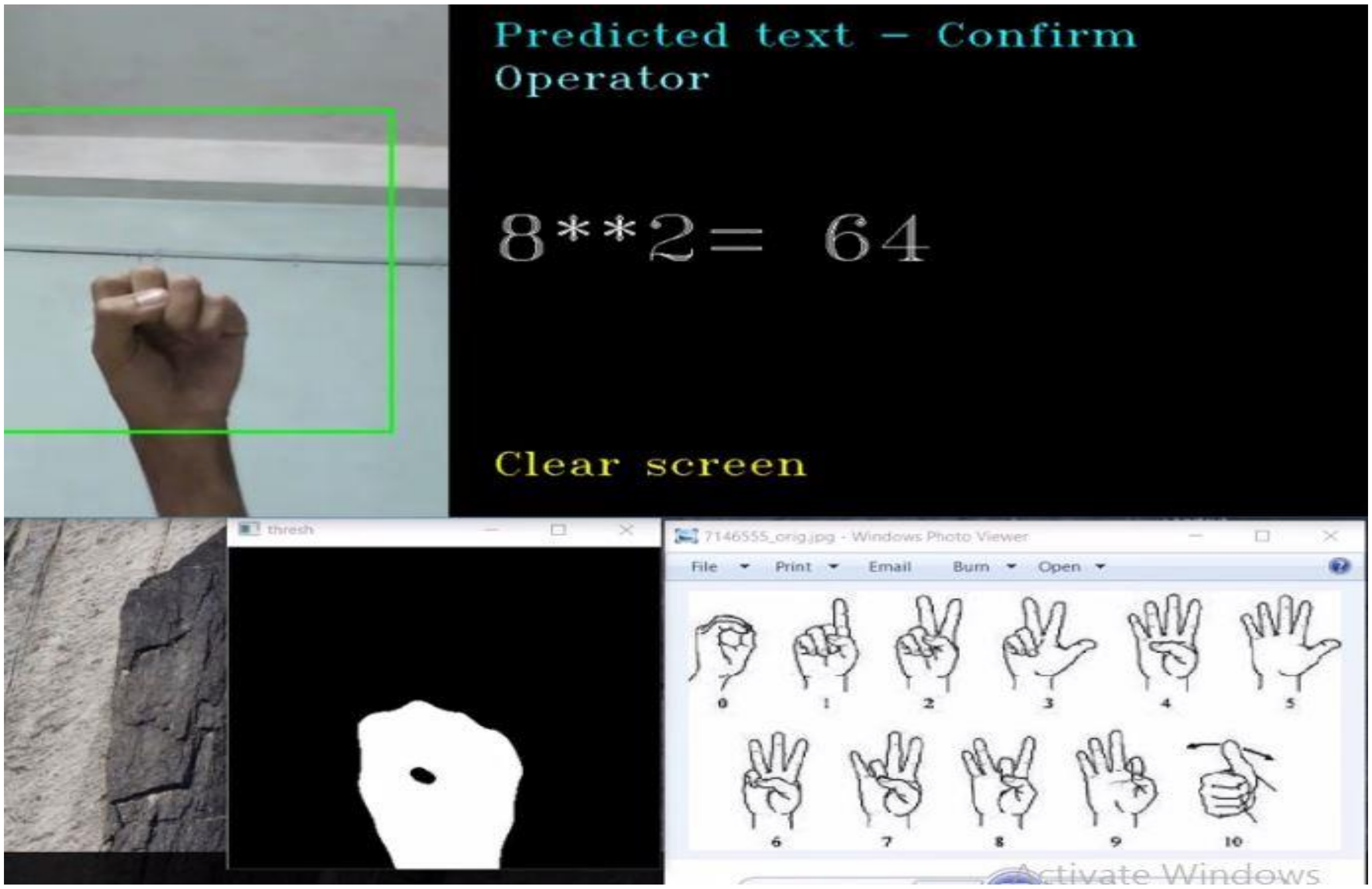
thresh
7146555_orig.jpg - Windows Photo Viewer
File Print Email Burn Open
0
1
2
3
4
5
6
7
8
9
10


Predicted text - Confirm
Operator

567*987= 559629

Clear screen

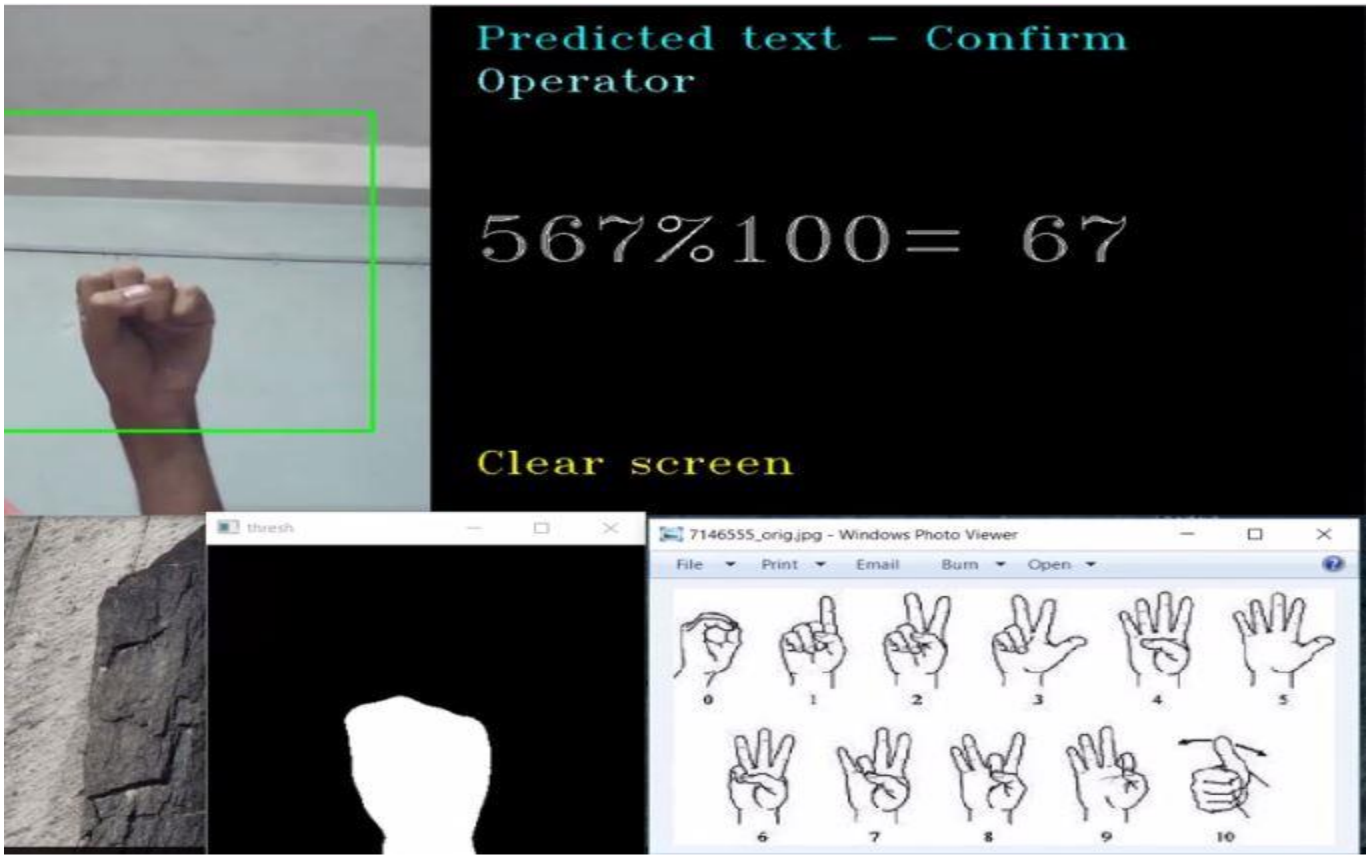
thresh
7146555_orig.jpg - Windows Photo Viewer
File Print Email Burn Open
0
1
2
3
4
5
6
7
8
9
10


Predicted text - Confirm
Operator

8**2= 64

Clear screen

thresh
7146555_orig.jpg - Windows Photo Viewer
File Print Email Burn Open
0
1
2
3
4
5
6
7
8
9
10
Activate Windows

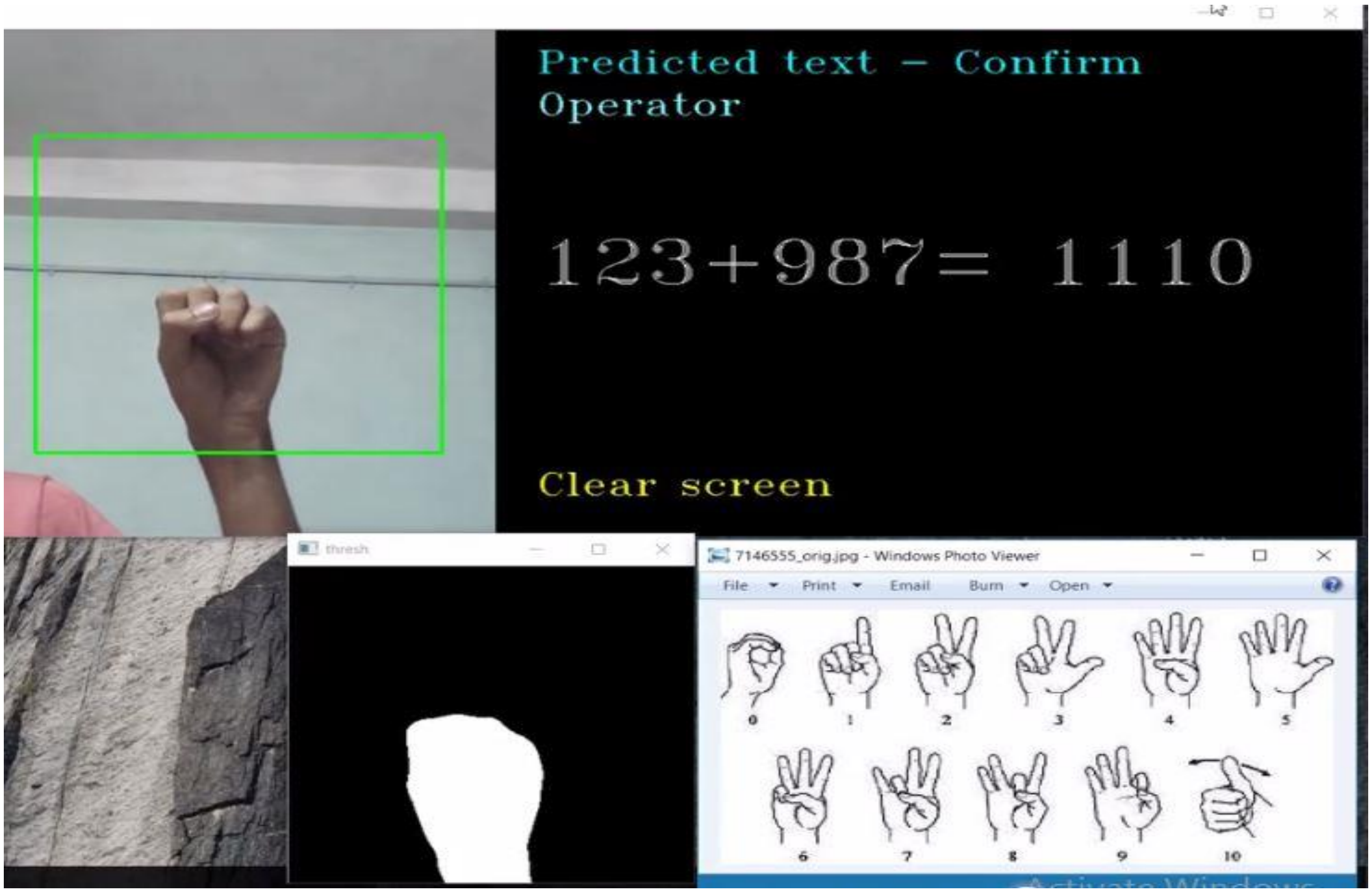


Predicted text - Confirm Operator

567%100= 67

Clear screen

thresh
7146555_orig.jpg - Windows Photo Viewer
File Print Email Burn Open
0
1
2
3
4
5
6
7
8
9
10

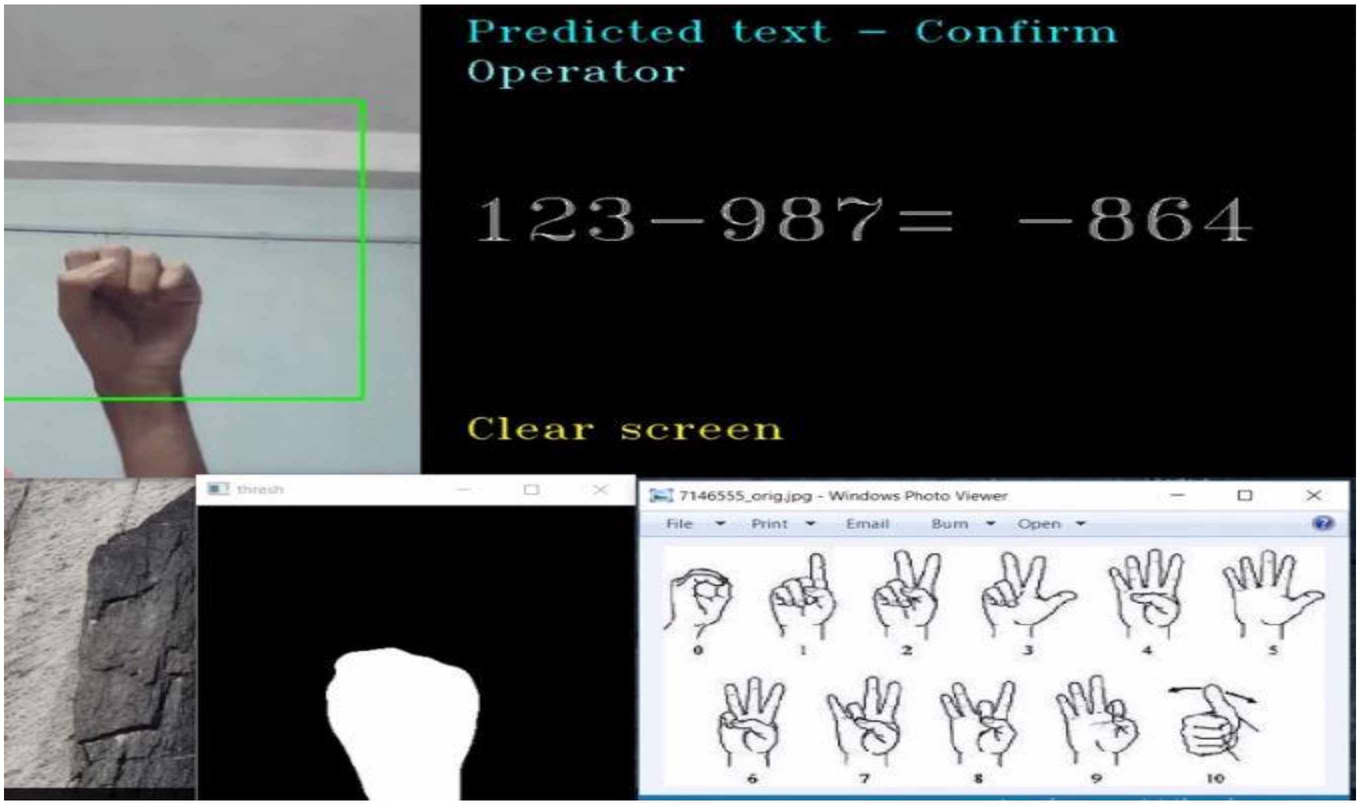


Predicted text - Confirm Operator

123+987= 1110

Clear screen

thresh
7146555_orig.jpg - Windows Photo Viewer
File Print Email Burn Open
0
1
2
3
4
5
6
7
8
9
10



❖ References:

[1] Research Article on Real-Time Hand Gesture Recognition Using Finger Segmentation by Zhi-hua Chen, Jung-Tae Kim, Jianning Liang, Jing Zhang, and Yu-Bo Yuan. Hindawi Publishing Corporation, The Scientific World Journal, 2014.

[2] Analyzing and Segmenting Finger Gestures in Meaningful Phases by Christos Mousas, Paul Newbury, Christos Nikolaos Anagnostopoulous, 2014 11th International Conference on Computer Graphics, Imaging and Visualization.

[3] Segmentation of the Character String Pointed Out by a Finger Tip from a Pair of Color Images by Shinji Tsuruoka, Makoto Nishikawa, Xinkai Chen, Muneaki Ishida from Department of Electrical and Electronic Engineering, Faculty of Engineering, Mie University, Japan, 1998.

[4] Efficient Finger Segmentation Robust to Hand Alignment in Imaging with Application to Human Verification by R.R.O. Al-Nima, S.S. Dlay, W.L. Woo and J.A. Chambers, 2017.

[5] Stack Overflow

[6] OpenCV.org