NAAN

MUDHALVAN –

IBM CLOUD BASED

PROJECT

TITLE : THE

GESTURES

RECOGNITION

FOR SIGN LANGUAGE(PHASE-

4) DEVELOPMENT PART 2

TEAM:7

1)K.RAJESH(T.L) - 422421106023

2)A.ALWIN XAVIER - 422421106003

3)T.DHIVAGAR - 422421106009

4)E.SURYA PRAKASH - 422421106029

Abstract

Our is to aim produce a model that can recognize hand gestures and signs. We will train a model for the purpose of sign language conversion, a simple gesture recognizing model; this will help people converse with people who are innately deaf and mentally disabled. This project can be implemented in several ways such as KNN, Logistic Regression, Naïve Bayes Classification, Support vector machine and can be implemented with CNN. The method we have chosen is CNN as it gives better accuracy compared to the rest of the methods. A computer program is developed using python language which is used to train the model based on the CNN algorithm. The program will be able to recognize hand gestures by comparing the input with preexisting dataset formed using the American sign Language. We will be able to convert Sign Language into text as output for users to recognize the signs presented by the sign language speaker. This model is implemented in Jypter Lab, an extension to the platform Anaconda documentation. To further improve, we will also add / integrate the inputs into black and white and take input from camera after using the method of Background subtraction. With the mask set to detect the human skin, this model will not require a plain background to function and can be implemented using a basic camera and a computing device.

Keywords : American sign recognition ; CNN ; background subtraction ; OpenCV ; hand tracking and segmentation ; feature extraction

1. Introduction

In the modern world, we have computers that work at very high speeds, making significantly huge amount of calculations in a split second. Now we humans are trying to achieve a goal where we want the computer to start thinking [1] and working like a human being. This requires the most basic property 'learning'. This takes us to artificial intelligence. According to AI, the computer starts or begins performing tasks on its own without human intervention. For this to happen, the computer needs to learn how to react to certain inputs and situations. The computer needs to be trained with huge amounts of data; the data that is used to train depends upon the preferred outcome and the working of the machine. We develop a computer model that can detect hand gestures made by human beings; there are so many applications that we see in our day to day life that work on hand gestures. Take a look at the console in our living room; hook it up with a sensor and we can start playing tennis with our hand. We realize a gesture recognition model that converts sign language into speech [2]. There are so many devices that exist that depend on gesture recognition, such as for security or entertainment purposes. Sign language may be a vision-based language that uses an amalgamation of various types of visuals, gestures, and the shape of the hands, fingers and their alignment, orientation, and movement of hand and body, eyes, lip, and complete facial movements and expressions. Like the spoken language, regional variants of signing also exist, e.g., Indian signing (ISL), American Sign Language (ASL), and Portuguese Sign Language, spelling each alphabet with our fingers, particular vocabulary is maintained for words and sentences, with the help of hands and body movement, facial expressions, and lip movement. Sign language can be isolated also as continuous. In isolated sign language, people communicate with gestures that represent a single word, while continuous sign language may be a sequence of gestures that generate a meaningful sentence [3]. All the methods for recognizing hand gestures are often broadly classified as vision-based and based on measurements picked up by sensors inside gloves. This vision-based method involves interaction between human and computer for gesture recognition, while the latter (glove based) method depends on the use of programmed external hardware for gesture recognition [4]. We will not be using a glove for our project; our model can recognize gestures made by bare hands. In this project, sign language recognition is done using OpenCV. It recognizes the hand gestures made by the user via a webcam; the output will be the text displayed on the screen. Our project aims to help people who are not aware of sign language by detecting the symbols made by a person and convert them into readable texts. We can achieve this with the help of Machine learning, specifically CNN [5]. We aim to train a model that predicts the text by taking an image as an input from the web camera/phone camera (using IP camera software and OpenCV). This model will be able to function at an accuracy of at least 75%, as the model is trained using a renowned dataset (ASL), enough data is available for the training algorithm to produce a precise and accurate model. The code used to develop the model is written in Python language, a simple computer can be used to realize the project without the need for high processing units and GPU's [6]. A platform called Anaconda Documentation is used as a base for the software Jypter Lab, where this model is trained and implemented [7]. The various concepts involved and the procedure on how this project is carried out is discussed in a detailed fashion in the upcoming sections of the paper.

1.1. Design and Description

OpenCV- We use OpenCV in Artificial Intelligence, face recognition, machine learning, etc. Computer Vision (CV) is an open-source Library in Python [8]. Computer Vision, helps the computer study objects through its eyes, sensors that allow the computer to take in visual data. Content of digital images such as videos and photographs are taken in by the machine through this platform. The objective of computer vision is to understand and extract data from the images [9]. It extracts the description from the pictures. It can be an object, a text description, and may also be a three-dimension model, and so on. For example, gaming consoles can be facilitated with computer vision, which will be able to identify different gestures made by the player with his arms and legs. And depending upon the movements made by the user, the computer performs the designated task in the game. The computer performance keeps on improving based on the time the data model is set to use on a single subject (one frequent user) signs, and so on, and acts accordingly [10]. Now machines with the help of computer vision has more of an edge to start thinking and acting like humans to some extent and can perform tasks like humans or similar efficiency. There are two tasks involved, object classification and object identification. In object classification, we train a model on a dataset of certain pre-defined objects; the model classifies these new objects arranging them to one or more of your training categories. In object identification, the object is identified based on its unique features which are in turn belonging to a certain class. OpenCV is used for computer Vision for the following reasons:

1) OpenCV is available free of cost.

2) Since the OpenCV library is written in C/C++, it is quite fast and can be used with Python.

3) It requires less RAM to usage, around 60-70 MB.

4) Computer Vision is as portable as OpenCV and can run on any device that C can run on.

Convolutional Neural Networks- Our entire model cannot be realized without this concept. CNN comes under deep neural networks belonging to one of its classes. We can see CNN applied in many areas but most of these areas including visual images. CNN is made up of several neurons whose value i.e., weights can be altered to serve the required purpose. These weights and biases are learnable [11]. Dot products are performed between the neurons resulting in non-linearity. The entire network expresses a single function; the main difference between the ordinary neural networks and convolutional neural networks is the assumption made by CNN, that all the inputs are explicit images. Thus, this will be best for training the model since our project revolves around images. Due to this, CNN will be able to take advantage the architecture can be constrained in a sensible way with images as input explicitly; an arrangement of neurons is done in 3 dimensions: width, depth and height. Depth means the volume required for activation.

Figure 1.

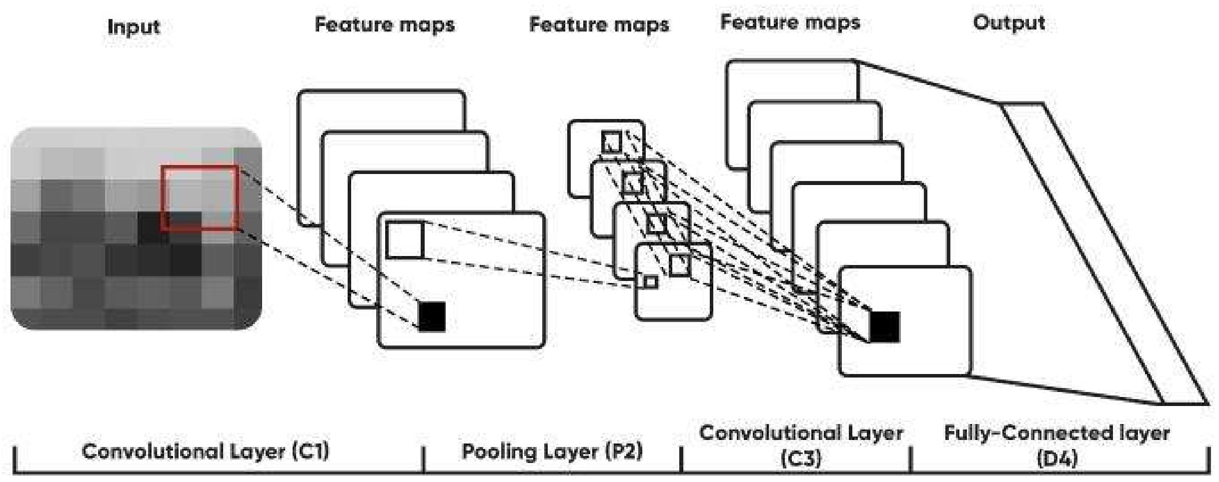


Figure 1. Structure of Convolutional Neural Network

As shown in Figure 1, we know that a convolutional network consists of a number of layers and the volume existing is transferred to another form with the help of a function(differentiable). The layers used to build CNN architecture are: pooling layer, convolutional layer and fully-connected layer. Stacking these in proper order will result in a convolutional network architecture [12]. Now these layers extract features that are unique to a property from the image, the loss function needs to be minimized. With this equation. it can be done so positive classes of the sample are denoted by M. Each positive class's CNN score is denoted by Sp; the scaling factor is denoted by 1/M

(1)

Background Subtraction- This means subtracting the background, i.e., taking it out of the picture. This technique involves separating the foreground with background elements. This is achieved with the help of a mask that is generated as per the user's desire [13]. We use this procedure to detect through static cameras. Background subtraction is vital for tracking an object, this can be realized in many ways.

1.2. How Does the Computer Recognize the Image

There is a lot of information around us and this is picked up by our eyes selectively, which is different for each person depending upon their preference. But the machines, on the other hand, see everything and take in every image and then convert the data into 1's and 0's which the computer can understand. How does this conversion take place? Pixel. This is the smallest unit of a digital image, which can be displayed or projected onto a display device. The image has various intensity levels at various positions and these levels are represented by numbers, for our images we have shown values consisting of only one value(grayscale), the value is assigned automatically by the computer based on the intensity of the darkness or level [14]. The two common ways to identify images are by Grayscale or RGB. In grayscale, picture a black and white image; these images consist of only two colors. The black color is assumed to be or treated as a measurement as the weakest intensity meaning white is the strongest intensity. The values are allotted by the computer based on the darkness levels [15]. In RGB, Red, green, blue are termed as RGG. All these colors together make an actual color. Any color that exists on this planet can be defined only using these three colors. The computer checks and extracts value from each pixel and assigns the results in an array.

Figure 2.

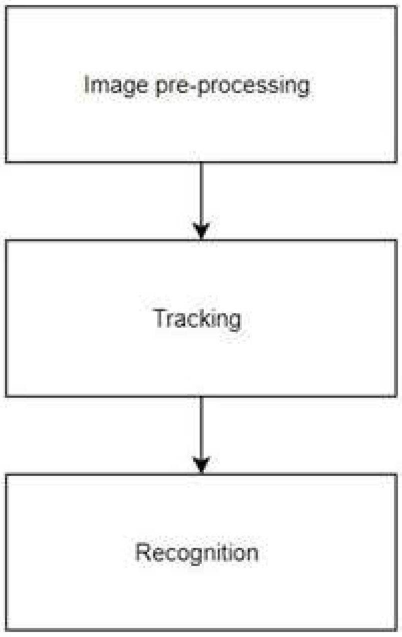


Figure 2. Gesture Recognition System

2. Proposed Methodology

As shown in Figure 2, we follow a process to devise a model that uses CNN to predict the text of a hand gesture/symbol. We also use methods like background subtraction (see Figure 3) to improve the efficiency of the model by eliminating the light/ambience of the background.

Figure 3.

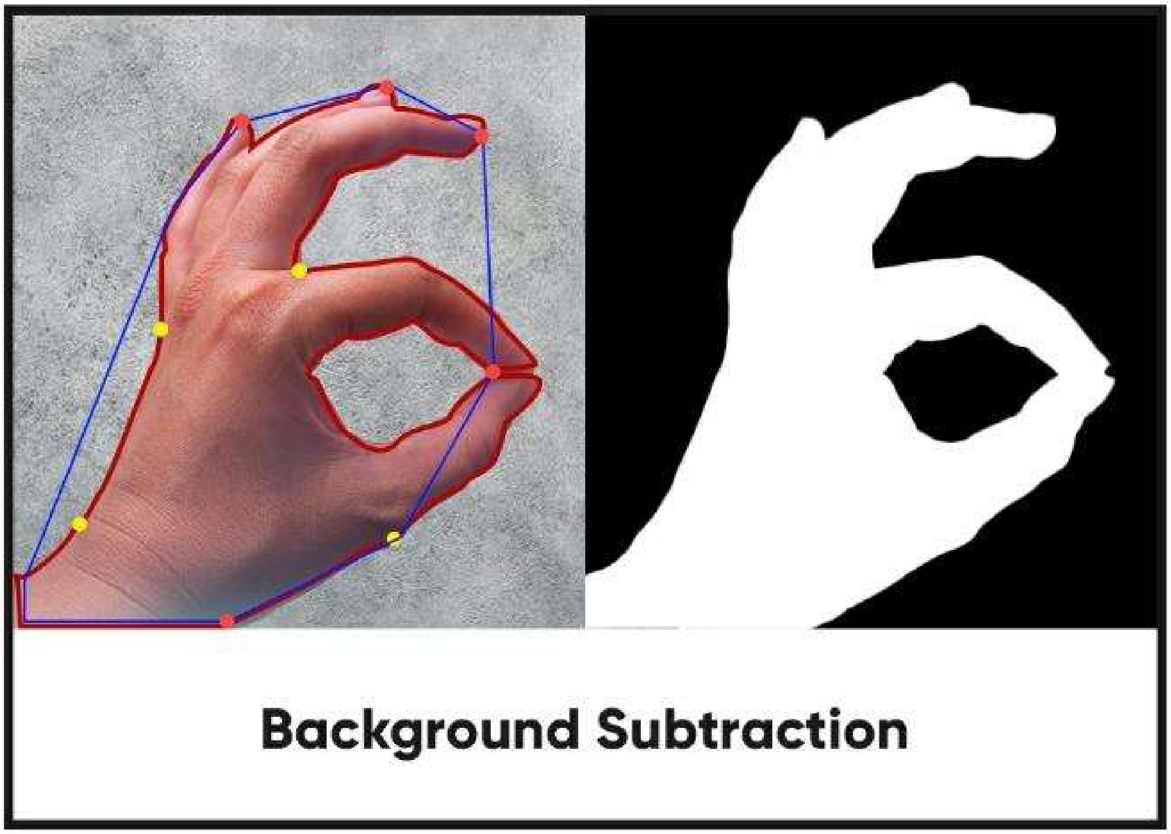


Figure 3. Background Subtraction

To build a Sign Language Recognition system, three things are needed:

1) Dataset as shown in Figure 4

2) Model (In this case we will use a CNN)

3) Platform to apply our model (We are going to use OpenCV)

Figure 4.

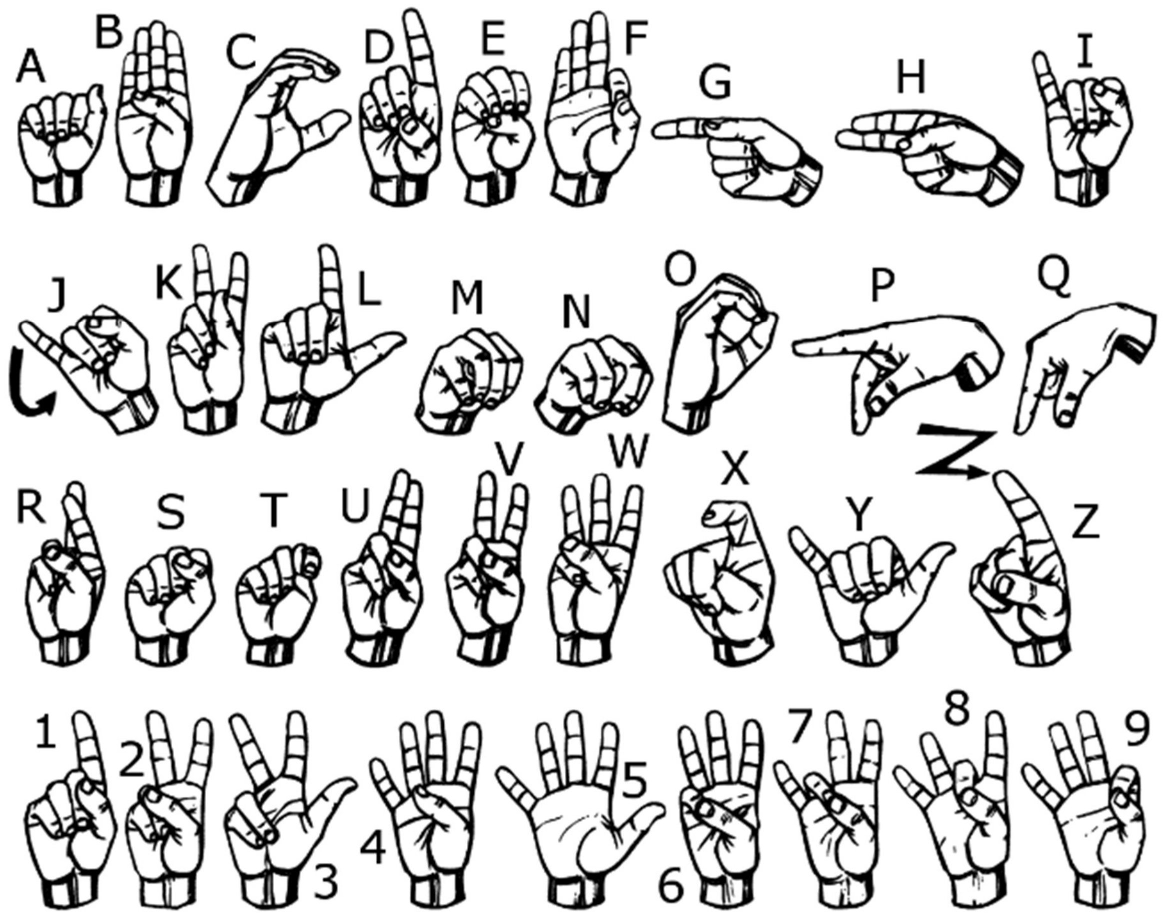


Figure 4. American Sign Language [ASL benchmark dataset]

To train a deep neural network, proper hardware is required (e.g., a powerful GPU). A powerful GPU will not be needed for this project. But still, the best way to go through with this is by using online platforms like Google Collab. We will use the Modified National Institute of Standards and Technology dataset [16]. Our dataset consists of many images of 24 (except J and Z) American Sign Language alphabet; there is a total of 784 pixels per image, meaning each image has a size of 28x28.

Our dataset is in CSV (Comma-separated values) format; train_X and test_Y contain the pixel values. train_Y and test_Y contains the label of image. Moving on to pre-processing, both the trained x and y consists an array of all the pixel values. An image is to be created from these values. Our image size is 28x28; we divide the array into 28x28 pixel groups. We use this dataset to coach our model. We use CNN (Convolutional Neural Network) to recognize the alphabets. We use keras. Like any other CNN, our model consists of a couple of layers like Conv2D and the MaxPooling that is followed by fully connected layers. The first Conv2D layer takes the shape of input image and the last fully connected layer provides us with the output for 26 alphabets [17]. We are using a Dropout after 2nd Conv2D layer to regularize our training. Soft Max is used as the activation function in the final layer [18], which will give us the probability for each alphabet as an output. The proposed model is given Table 1.

Table 1. Model Overview

Layer (Type)	Output Shape	Param#
Conv2d_1(Conv2D)	(None,28,8,8)	80
Max_pooling_1(MaxPooling2)	(None,14,14,8)	0
Conv2d_2(Conv2D)	(None,14,14,16)	1168
Dropout_1(Dropout)	(None,14,14,16)	0
Max_pooling2d_2(MaxPooling2)	(None,3,3,16)	0
Dense_1(dense)	(None,3,3,128)	2176
Flatten_1(Flatten)	(None,1152)	0
Dense_2(dense)	(none,26)	29978

New window | CSV

The SGD optimizer is utilized for compiling our model. You may decrease the epochs to 25. Check the accuracy of the trained model. Open CV: We must create a window to take the input from our webcam. The images that are taken as input should be a grayscale image(28x28). Because we trained our model on 28x28 size image, the alphabet from the input image is to be predicted. Our model will give outputs as integers rather than alphabets; that is because the labels are given as integers (1-A, 2-B, 3-C, etc.). With maximum value of accuracy, our model should be able to recognize the alphabet, with the help plain background and descent lights, without any hindrance.

2.1. Steps Inolved

Hand Recognition: I window is required to take the input image from our camera (web cam) so we must first create it. As the requirements of the image dimension mentioned earlier(28x28), a greyscale image of these dimensions must be taken. Because the model has been trained for those dimensions only, following the same dimensions will result in accurate results. Segmentation: After tracking the hands, the next step is to segment the hands from the background. The HSV color model is used for this purpose. Feature Extraction: Several general features are extracted; the relationship between features and classes is inferred by an appropriate classifier. Gesture Recognition: After extracting features of the input character, we search its features in the database and consider the most similar features as the result.

2.2. Execution of the Model

The execution includes the following steps:

·Capture the image through the camera.

·Before sending the captured image to the server, store some of the test images of the person.

·In the server, give it to the learning API.

·Now send this captured image through the API to the server.

·To send this image to the server we use REST API

CODE:

```
#!/usr/bin/env python

import warnings

warnings.filterwarnings("ignore")


"""

IMPORTS

"""

from skimage import data, io, filters

from skimage.transform import rescale

from skimage.color import rgb2gray

from skimage.feature import hog

from skimage.transform import resize


import pickle

import numpy as np

import pandas as pd


import glob

import random

import csv
```

```
from os import listdir

from os.path import isfile, join


from sklearn.ensemble import  RandomForestClassifier


from sklearn.preprocessing import LabelEncoder

from sklearn.svm import LinearSVC

from sklearn.naive_bayes import MultinomialNB,GaussianNB

from sklearn.linear_model import LogisticRegression

from sklearn.svm import SVC


import json

import time

import gzip


#given a list of filenames return s a dictionary of images

def getfiles(filenames):

    dir_files = {}

    for x in filenames:

        dir_files[x]=io.imread(x)

    return dir_files
```

```
#return hog of a particular image vector
def convertToGrayToHOG(imgVector):
    rgbImage = rgb2gray(imgVector)
    return hog(rgbImage)


#takes returns cropped image
def crop(img,x1,x2,y1,y2):
    crp=img[y1:y2,x1:x2]
    crp=resize(crp,((128,128)))#resize
    return crp


#save classifier
def dumpclassifier(filename,model):
    with open(filename, 'wb') as fid:
        pickle.dump(model, fid)


#load classifier
def loadClassifier(picklefile):
    fd = open(picklefile, 'r+')
    model = pickle.load(fd)
    fd.close()

    return model


"""

This function randomly generates bounding boxes

Return: hog vector of those cropped bounding boxes along with label

Label : 1 if hand ,0 otherwise

"""

def buildhandnothand_lis(frame,imgset):

    poslis =[]

    neglis =[]


    for nameimg in frame.image:

        tupl = frame[frame['image']==nameimg].values[0]

        x_tl = tupl[1]

        y_tl = tupl[2]

        side = tupl[5]

        conf = 0


        dic = [0, 0]
```

```
            arg1 = [x_tl,y_tl,conf,side,side]

poslis.append(convertToGrayToHOG(crop(imgset[nameimg],x_tl,x_tl+side,y_tl,y_tl
+side)))
            while dic[0] <= 1 or dic[1] < 1:
               x = random.randint(0,320-side)
               y = random.randint(0,240-side)
               crp = crop(imgset[nameimg],x,x+side,y,y+side)
               hogv = convertToGrayToHOG(crp)
               arg2 = [x,y, conf, side, side]


               z = overlapping_area(arg1,arg2)
               if dic[0] <= 1 and z <= 0.5:
                  neglis.append(hogv)
                  dic[0] += 1
               if dic[0]== 1:
                  break
        label_1 = [1 for i in range(0,len(poslis)) ]
        label_0 = [0 for i in range(0,len(neglis))]
        label_1.extend(label_0)
        poslis.extend(neglis)
        return poslis,label_1
```

```
    #returns imageset and bounding box for a list of users
    def train_binary(train_list, data_directory):
        frame = pd.DataFrame()
        list_ = []
        for user in train_list:

list_.append(pd.read_csv(data_directory+user+'/'+user+'_loc.csv',index_col=None,
header=0))
        frame = pd.concat(list_)
        frame['side']=frame['bottom_right_x']-frame['top_left_x']
        frame['hand']=1

        imageset = getfiles(frame.image.unique())

        #returns actual images and dataframe
        return imageset,frame

    #loads data for binary classification (hand/not-hand)
    def load_binary_data(user_list, data_directory):
        data1,df  =train_binary(user_list, data_directory) # data 1 - actual images , df
is actual bounding box
```

```
    # third return, i.e., z is a list of hog vecs, labels

    z = buildhandnothand_lis(df,data1)

    return data1,df,z[0],z[1]




#loads data for multiclass

def get_data(user_list, img_dict, data_directory):

    X = []

    Y = []


    for user in user_list:

        user_images = glob.glob(data_directory+user+'/*.jpg')


        boundingbox_df = pd.read_csv(data_directory+user+'/'+user+'_loc.csv')


        for rows in boundingbox_df.iterrows():

            cropped_img = crop(img_dict[rows[1]['image']], rows[1]['top_left_x'],
rows[1]['bottom_right_x'], rows[1]['top_left_y'], rows[1]['bottom_right_y'])

            hogvector = convertToGrayToHOG(cropped_img)

            X.append(hogvector.tolist())

            Y.append(rows[1]['image'].split('/')[1][0])
    return X, Y


#utility funtcion to compute area of overlap

def overlapping_area(detection_1, detection_2):

    x1_tl = detection_1[0]

    x2_tl = detection_2[0]

    x1_br = detection_1[0] + detection_1[3]

    x2_br = detection_2[0] + detection_2[3]

    y1_tl = detection_1[1]

    y2_tl = detection_2[1]

    y1_br = detection_1[1] + detection_1[4]

    y2_br = detection_2[1] + detection_2[4]

    # Calculate the overlapping Area

    x_overlap = max(0, min(x1_br, x2_br)-max(x1_tl, x2_tl))

    y_overlap = max(0, min(y1_br, y2_br)-max(y1_tl, y2_tl))

    overlap_area = x_overlap * y_overlap

    area_1 = detection_1[3] * detection_2[4]

    area_2 = detection_2[3] * detection_2[4]

    total_area = area_1 + area_2 - overlap_area

    return overlap_area / float(total_area)
```

```
"""
Does hard negative mining and returns list of hog vectos , label list and no_of_false_positives after sliding
"""
def do_hardNegativeMining(cached_window,frame, imgset, model, step_x, step_y):
    lis = []
    no_of_false_positives = 0
    for nameimg in frame.image:
        tupl = frame[frame['image']==nameimg].values[0]
        x_tl = tupl[1]
        y_tl = tupl[2]
        side = tupl[5]
        conf = 0

        dic = [0, 0]

        arg1 = [x_tl,y_tl,conf,side,side]
        for x in range(0,320-side,step_x):
            for y in range(0,240-side,step_y):
                arg2 = [x,y,conf,side,side]
                z = overlapping_area(arg1,arg2)
                prediction                                                          =
model.predict([cached_window[str(nameimg)+str(x)+str(y)]])[0]

                if prediction == 1 and z<=0.5:
                    lis.append(cached_window[str(nameimg)+str(x)+str(y)])
                    no_of_false_positives += 1

        label = [0 for i in range(0,len(lis))]
        return lis,label, no_of_false_positives

    """
    Modifying to cache image values before hand so as to not redo that again and again

    """
    def cacheSteps(imgset, frame ,step_x,step_y):
        # print "Cache-ing steps"
        list_dic_of_hogs = []
        dic = {}
        i = 0
        for img in frame.image:
            tupl = frame[frame['image']==img].values[0]

            x_tl = tupl[1]

            y_tl = tupl[2]

            side = tupl[5]

            conf = 0

            i += 1

            # if i%10 == 0:

            #     print "{0} images cached ".format(i)

            imaage = imgset[img]

            for x in range(0,320-side,step_x):

                for y in range(0,240-side,step_y):

dic[str(img+str(x)+str(y))]=convertToGrayToHOG(crop(imaage,x,x+side,y,y+side))

        return dic




    def improve_Classifier_using_HNM(hog_list, label_list, frame, imgset,
threshold=50, max_iterations=25): # frame - bounding boxes-df; yn_df - yes_or_no
df

        # print "Performing HNM :"

        no_of_false_positives = 1000000     # Initialise to some random high value

        i = 0
        step_x = 32

        step_y = 24


        mnb  = MultinomialNB()

        cached_wind = cacheSteps(imgset, frame, step_x, step_y)


        while True:

            i += 1

            model = mnb.partial_fit(hog_list, label_list, classes = [0,1])


            ret = do_hardNegativeMining(cached_wind,frame, imgset, model,
step_x=step_x, step_y=step_y)


            hog_list = ret[0]

            label_list = ret[1]

            no_of_false_positives = ret[2]


            if no_of_false_positives == 0:

                return model
        print "Iteration {0} - No_of_false_positives: {1}".format(i,
no_of_false_positives)

        if no_of_false_positives <= threshold:
            return model

        if i>max_iterations:
             return model

    # Malisiewicz et al.
    def non_max_suppression_fast(boxes, overlapThresh):
        # print "Perfmorinf NMS:"
        # if there are no boxes, return an empty list
        if len(boxes) == 0:
            return []

        # if the bounding boxes integers, convert them to floats --
        # this is important since we'll be doing a bunch of divisions
        if boxes.dtype.kind == "i":
            boxes = boxes.astype("float")

        # initialize the list of picked indexes
```

```
pick = []


# grab the coordinates of the bounding boxes
x1 = boxes[:,0]
y1 = boxes[:,1]
x2 = boxes[:,2]
y2 = boxes[:,3]
s = boxes[:,4]
# compute the area of the bounding boxes and sort the bounding
# boxes by the bottom-right y-coordinate of the bounding box
area = (x2 - x1 + 1) * (y2 - y1 + 1)
idxs = np.argsort(s)


# keep looping while some indexes still remain in the indexes
# list
while len(idxs) > 0:
    # grab the last index in the indexes list and add the
    # index value to the list of picked indexes
    last = len(idxs) - 1
    i = idxs[last]
    pick.append(i)
```

```
        # find the largest (x, y) coordinates for the start of

        # the bounding box and the smallest (x, y) coordinates

        # for the end of the bounding box

        xx1 = np.maximum(x1[i], x1[idxs[:last]])

        yy1 = np.maximum(y1[i], y1[idxs[:last]])

        xx2 = np.minimum(x2[i], x2[idxs[:last]])

        yy2 = np.minimum(y2[i], y2[idxs[:last]])


        # compute the width and height of the bounding box

        w = np.maximum(0, xx2 - xx1 + 1)

        h = np.maximum(0, yy2 - yy1 + 1)


        # compute the ratio of overlap

        overlap = (w * h) / area[idxs[:last]]


        # delete all indexes from the index list that have

        idxs = np.delete(idxs, np.concatenate(([last],

            np.where(overlap > overlapThresh)[0])))


    # return only the bounding boxes that were picked using the
    # integer data type

    return boxes[pick].astype("int")


# Returns the tuple with the highest prediction probability of hand

def image_pyramid_step(model, img, scale=1.0):

    max_confidence_seen = -1

    rescaled_img = rescale(img, scale)

    detected_box = []

    side = 128

    x_border = rescaled_img.shape[1]

    y_border = rescaled_img.shape[0]


    for x in range(0,x_border-side,32):

        for y in range(0,y_border-side,24):

            cropped_img = crop(rescaled_img,x,x+side,y,y+side)

            hogvector = convertToGrayToHOG(cropped_img)


            confidence = model.predict_proba([hogvector])


            if confidence[0][1] > max_confidence_seen:

                detected_box = [x, y, confidence[0][1], scale]
                max_confidence_seen = confidence[0][1]

        return detected_box

    """
    ===============================================================
==================================================================
=============
    """

    class GestureRecognizer(object):
        """class to perform gesture recognition"""

        def __init__(self, data_director='./'):
            """
                data_directory : path like /home/sanket/mlproj/dataset/
                includes the dataset folder with '/'
                Initialize all your variables here
            """
            self.data_directory = data_director
            self.handDetector = None
            self.signDetector = None
            self.label_encoder = LabelEncoder().fit(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K',
'L', 'M', 'N',
            'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y'])


        def __init__(self,data_dir, hand_Detector, sign_Detector):
            self.data_directory = data_dir
            self.handDetector = loadClassifier(hand_Detector)
            self.signDetector = loadClassifier(sign_Detector)
            self.label_encoder = LabelEncoder().fit(['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I', 'K',
'L', 'M', 'N',
            'O', 'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y'])



        def train(self, train_list):
            """
                train_list : list of users to use for training
                eg ["user_1", "user_2", "user_3"]
                The train function should train all your classifiers
                both binary and multiclass on the given list of users
            """
            print "Train starts"
            # Load data for the binary (hand/not hand) classification task
            imageset, boundbox, hog_list, label_list = load_binary_data(train_list,
self.data_directory)

            print "Imageset, boundbox, hog_list,label_list Loaded!"

            # Load data for the multiclass classification task
            X_mul,Y_mul = get_data(train_list, imageset, self.data_directory)

            print "Multiclass data loaded"



            Y_mul = self.label_encoder.fit_transform(Y_mul)

            if self.handDetector == None:
                # Build binary classifier for hand-nothand classification
                self.handDetector = improve_Classifier_using_HNM(hog_list, label_list,
boundbox, imageset, threshold=40, max_iterations=35)

            print "handDetector trained "

            # Multiclass classification part to classify the various signs/hand gestures
CHECK. TODO.
```

```
    if self.signDetector == None:

        svcmodel = SVC(kernel='linear', C=0.9, probability=True)

        self.signDetector = svcmodel.fit(X_mul, Y_mul)



    print "sign Detector trained "


    # dumpclassifier('handDetector.pkl', self.handDetector)


    # dumpclassifier('signDetector.pkl', self.signDetector)


    # dumpclassifier('label_encoder.pkl', self.label_encoder)



def recognize_gesture(self, image):

    """

        image : a 320x240 pixel RGB image in the form of a numpy array


        This function should locate the hand and classify the gesture.

        returns : (position, label)
    position : a tuple of (x1,y1,x2,y2) coordinates of bounding box
        x1,y1 is top left corner, x2,y2 is bottom right

    label : a single character. eg 'A' or 'B'
"""
# print "In recognize_gesture"
scales = [   1.25,
         1.015625,
         0.78125,
         0.546875,
         1.5625,
         1.328125,
         1.09375,
         0.859375,
         0.625,
         1.40625,
         1.171875,
         0.9375,
         0.703125,
         1.71875,
                1.484375
            ]

        detectedBoxes = [] ## [x,y,conf,scale]
        for sc in scales:

detectedBoxes.append(image_pyramid_step(self.handDetector,image,scale=sc))


        side = [0 for i in xrange(len(scales))]
        for i in xrange(len(scales)):
            side[i]= 128/scales[i]


        for i in xrange(len(detectedBoxes)):
            detectedBoxes[i][0]=detectedBoxes[i][0]/scales[i] #x
            detectedBoxes[i][1]=detectedBoxes[i][1]/scales[i] #y


        nms_lis = [] #[x1,x2,y1,y2]
        for i in xrange(len(detectedBoxes)):
            nms_lis.append([detectedBoxes[i][0],detectedBoxes[i][1],

detectedBoxes[i][0]+side[i],detectedBoxes[i][1]+side[i],detectedBoxes[i][2]])
        nms_lis = np.array(nms_lis)
```

```
res = non_max_suppression_fast(nms_lis,0.4)


output_det = res[0]

x_top = output_det[0]

y_top = output_det[1]

side = output_det[2]-output_det[0]

position = [x_top, y_top, x_top+side, y_top+side]


croppedImage = crop(image, x_top, x_top+side, y_top, y_top+side)

hogvec = convertToGrayToHOG(croppedImage)


prediction = self.signDetector.predict_proba([hogvec])[0]


zi = zip(self.signDetector.classes_, prediction)

zi.sort(key = lambda x:x[1],reverse = True)


# To return the top 5 predictions

final_prediction = []

for i in range(5):

   final_prediction.append(self.label_encoder.inverse_transform(zi[i][0]))
```

```
    # print position,final_prediction

    return position,final_prediction

def save_model(self, **params):

    """
        save your GestureRecognizer to disk.
    """

    self.version = params['version']
    self.author = params['author']

    file_name = params['name']

    pickle.dump(self, gzip.open(file_name, 'wb'))
    # We are using gzip to compress the file
    # If you feel compression is not needed, kindly take lite

@staticmethod       # similar to static method in Java
def load_model(**params):
        """
            Returns a saved instance of GestureRecognizer.


            load your trained GestureRecognizer from disk with provided params
            Read - http://stackoverflow.com/questions/36901/what-does-double-star-and-star-do-for-parameters
        """


        file_name = params['name']
        return pickle.load(gzip.open(file_name, 'rb'))


        # People using deep learning need to reinitalize model, load weights here etc.


    # def main():
    #     #handDetector = loadClassifier('./handDetector.pkl')
    #     #signDetector = loadClassifier('./signDetector.pkl')
    #     # self,data_dir,hand_Detector,sign_Detector
    #         gs = GestureRecognizer('/home/ml/Suks_ml_project/dataset/', 'model_log_2.pkl', 'signDetector.pkl')
    #         userlist=[ 'user_11', 'user_12','user_13','user_14','user_15','user_16','user_17','user_18','user_19','user_3'
,
    #          'user_4','user_5','user_6','user_7','user_9','user_10']


    #     user_tr = userlist[:1]

    #     user_te = userlist[-1:]

    #     gs.train(user_tr)


    #    gs.save_model(name = "sign_detector.pkl.gz", version = "0.0.1", author = 'ss')



    #     print "The GestureRecognizer is saved to disk"


    #     new_gr = GestureRecognizer.load_model(name = "sign_detector.pkl.gz") #
automatic dict unpacking

    #     print new_gr.label_encoder

    #     print new_gr.signDetector





    # if __name__ == '__main__':

    #     main()
```

·REST stands for Representational State Transfer, meaning when a REST API is called.

·The server will transfer the representation of the state of the requested source to the client.

·After sending this image to the server, the server will finalize the results i.e., gives back the name and employee id of the person in that image using learning API and data sets to the REST API.

·REST API gives back the information passed by the server to the clients.

Figure 5.

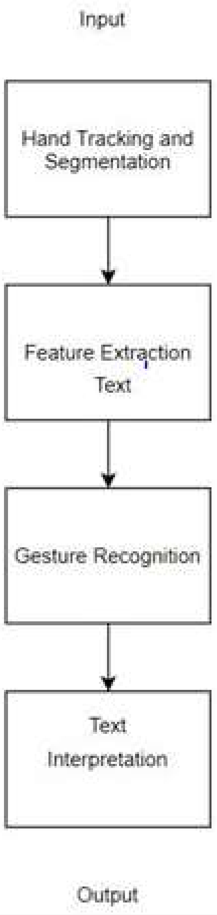
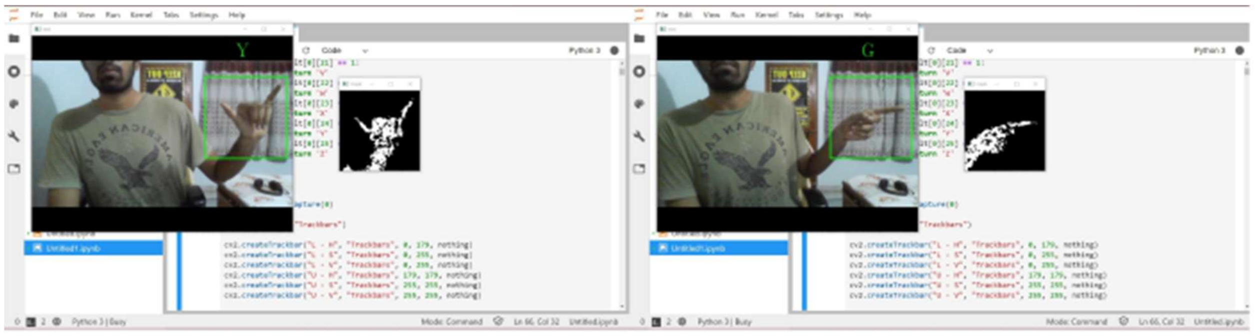


Figure 5. Execution Process

3. Outputs

After opening the platform to perform the execution of the model as shown in Figure 5, we used jupyter lab to go through with the execution of our model launched by anaconda navigator as the base software when the code is run. We are presented with three windows: mask window, the camera window and the trackbar window. It is advised to have a plain background before going through with the recognition process, but if that seems to be an issue we can always change the HSV colour scheme in the trackbar to match that ratio to that of the human skin. So even if the background is not plain or a solid color, we will be able to make the model work at good efficiency. I am using a basic dell model laptop and working from home bcause of the pandemic. Due to this, I do not have many resources at my expense. I decided to go through the model using a computer software which enables the built-in webcam to act as the input device. To resolve the background issue, I set the trackbar levels to upper HSV- 0,58,50 and lower HSV- 30,255,255. With this setting we will be able to deliver a black background where our skin will be detected in white. We move on to checking the accuracy and the precision of the model. As mentioned in Figure 3, our model can detect American sign language and display the respective alphabet or character in text on screen. I have tested the model and am displaying four random alphabets as detected by the computer. Parts of the source code are also visible in the background.

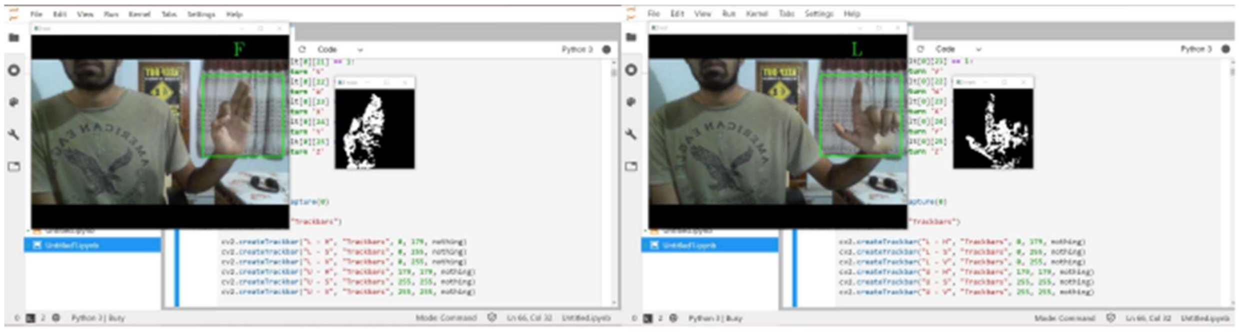
Figure 6.



```
cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U - V", "Trackbars", 255, 255, nothing)
```

Figure 6. Alphabets ‘Y’ and ‘G’ as detected by the model

Figure 7.



```
cv2.createTrackbar("L - H", "Trackbars", 0, 179, nothing)
cv2.createTrackbar("L - S", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("L - V", "Trackbars", 0, 255, nothing)
cv2.createTrackbar("U - H", "Trackbars", 179, 179, nothing)
cv2.createTrackbar("U - S", "Trackbars", 255, 255, nothing)
cv2.createTrackbar("U - V", "Trackbars", 255, 255, nothing)
```

Figure 7. Alphabets ‘F’ and ‘L’ as detected by the model

As seen in the output screenshots shown in Figure 6 and Figure 7, I did not use a plain background but due to the process of background subtraction and selecting the appropriate HSV values in the trackbar, we were able to secure a mask that can easily detected and distinguished human skin from other elements existing in the camera vision. This feature is achieved due to selection of proper algorithm to model the background and foreground separator.

4. Applications

·For ssecurity purposes, companies are using face recognition to secure their premises.

·Immigration checkpoints use facial recognition to enforce smarter border control.

·Vehicle security management companies can use facial recognition to give access to the engine and secure their vehicles.

·Ride-sharing companies can use facial recognition to ensure proper passengers are picked up by the proper drivers [19].

·IoT benefits from facial recognition allows enhanced security measures and automatic access control reception.

·Enforcement can use facial recognition technologies together as a part of AI-driven surveillance systems.

·Retailers can use facial recognition to customize offline offerings and to theoretically map online purchasing habits with their online ones [20].

5. Conclusion

The dimensionality of representations is often exploited to move representations to a higher level by discovering the spatial or spatiotemporal regions of interest or selecting/extracting features that enhance the discrimination of similar looking expressions of different emotions. To these ends, most existing systems rely on generic dimensionality reduction techniques. The optimality of such techniques, however, is being questioned in the scope of affect recognition, and new trends address the importance of making use of domain knowledge explicitly when developing dimensionality reduction technique.

In this survey, we analyzed facial recognition systems by breaking them down into their fundamental components and we analyzed their potentials and limitations. In this section, we summarize the progress in the literature and highlight future directions. Advances in the field and the transition from controlled to naturalistic settings have been the focus of several survey papers. Zeng et al. focused on automatic affect recognition using visual and auditory modalities. Gunes and Schuller highlighted the continuity aspect for affect recognition both in terms of input and system output. Yet no survey has analyzed systems by isolating their fundamental components and discussing how each component addresses the above-mentioned challenges in facial affect recognition. Furthermore, some new trends and developments are not discussed in previous survey papers. Novel classification techniques that aim at capturing affect specific dynamics are proposed; validation protocols with evaluation metrics tailored for affect analysis are presented and affect recognition competitions are organized. Our in-depth analysis of these developments will expose open issues and useful practices and facilitate the design of real-world affect recognition systems.

The dimensionality of representations is often exploited to move representations to a higher level by discovering the spatial or spatiotemporal regions of interest, or selecting/extracting features that enhance the discrimination of similar looking expressions of different emotions. To these ends, most existing systems rely on generic dimensionality reduction techniques. The optimality of such techniques, however, is being questioned in the scope of affect recognition, and new trends address the importance of making use of domain knowledge explicitly when developing dimensionality reduction technique