

# **DIFFERENT HAND GESTURE RECOGNITION USING ANN**

## **PROJECT REPORT**

*Submitted in fulfilment for the JComponent of  
Soft Computing (ITE1015)*

## **CAL Course**

*in*

**B.Tech. – Information Technology**

*by*

**SANYAM JAIN (20BIT0428)**

**TRISHIT DEVENDER GUPTA (20BIT0374)**

*Under the guidance of*

**Dr. S. Hemalatha**

**SITE**



**VIT<sup>®</sup>**  
**Vellore Institute of Technology**  
(Deemed to be University under section 3 of UGC Act, 1956)

**School of Information Technology and Engineering**

Winter Semester 2021-22

## TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO.
	<b>ABSTRACT</b>	
<b>1.</b>	<b>Introduction</b>	<b>3</b>
<b>2.</b>	<b>Dataset Specification</b>	<b>4</b>
<b>3.</b>	<b>Literature Review (all papers considered in a team)</b>	
	<b>3.1.Review on Various Schemes</b>	<b>5</b>
	<b>3.2.Comparative study on various subtitles</b>	<b>10</b>
<b>4.</b>	<b>System design</b>	
	<b>4.1.Architecture Diagram / Flow Diagram / Flowchart / ...</b>	<b>13</b>
	<b>4.2.Detailed Description of Modules</b>	<b>14</b>
<b>5.</b>	<b>Software Requirement Specifications</b>	<b>16</b>
<b>6.</b>	<b>Experimental Results &amp; Discussion</b>	
	<b>6.1.Source code</b>	<b>16</b>
	<b>6.2.Screenshots with Explanation</b>	<b>28</b>
<b>7.</b>	<b>References</b>	<b>39</b>
Annexure	<b>Attach PDF of main reference paper</b>	

## Introduction:

The necessity of Hand Gesture Recognition is having a well-defined dataset which have sufficient number of images for every hand gesture by which we can predict the hand gesture more efficiently. Hand Gesture Recognition idea begin with the invention of glove-based control interface for the computer control, then over many years this technology was upgraded and now we are able to have a smart touchless hand gesture recognition product which reduce human efforts indefinitely.

For Real-time problems like:

- i) Controlling a laptop/personal mobile using hand gesture recognition technique.
- ii) For the people who are deaf or dumb understanding the talk using hand gesture recognition technique.

For the controlling a laptop/personal mobile we had to make some fix hand notations by using which the user can perform that specific task, and by using notations which are the user make we can use CNN which can help us to predict the notation user wants to convey thereby for that we can have a dataset for the notations which can help our model to predict the notation more accurately, hence the computer/mobile can do the right operation which user wants. As, 3D-CNN can be useful for detecting a set of action (i.e., the action which require hand moments) which can make the system more efficient and hence provide us with many new gesture notations.

For people who are deaf or can't speak the hand gesture recognition can help others to understand their talk with hands and them to understand the other people's talk using hands. For this we can have some common actions which a person uses by expressing their feelings and based on that we can create some notations by which a general talk of normal people can also be understood by using this technique. This will let the disabled people to live a life like a normal human by understanding other's and making the other's understanding them.

As, we get the notations for the following cases we can create a dataset by including all the types of images a user can use to express a specific notation, then we can train the dataset using CNN or 3D CNN as in general a notation may have more than one actions so by using it, we can predict the notations and its meaning based on the user actions accurately.

Hand gestures are powerful human to human communication channel which convey a major part of information transfer in our everyday life. Hand gesture recognition is a process of understanding and classifying meaningful movements by the human hands.

We need a well defined dataset is the necessity which has images for hand gestures by which we can predict the hand gestures more efficiently.

The history of hand gesture recognition for computer control **started with the invention of glove-based control interfaces**. Researchers realized that gestures inspired by sign language can be used to offer simple commands for a computer interface.

### **Real Time Problems:-**

- i) In the automotive industry, this capability allows drivers and passengers to interact with the vehicle — usually to control the infotainment system without touching any buttons or screens.
- ii) Sign Language Interpreter for disabled people

- i) A gesture recognition system starts with a camera pointed at a specific three-dimensional zone within the vehicle, capturing frame-by-frame images of hand positions and motions. This camera is typically mounted in the roof module or other vantage point that is unlikely to be obstructed. The system illuminates the area with infrared LEDs or lasers for a clear image even when there is not much natural light. Those images are analysed in real time by computer vision and machine learning technologies, which translate the hand motions into commands, based on a predetermined library of signs.

- ii) **How does sign language recognition work?**

Identification of sign gesture is mainly performed by the following methods:

1. Glove-based method in which the signer has to wear a hardware glove, while the hand movements are getting captured.
2. Vision-based method, further classified into static and dynamic recognition.

### **Dataset Specification:**

We are going to use a dataset which contains images of many different types of gesture (namely one finger notation gesture, two finger notation gesture, left notation gesture, etc.) and around 1000 images for each gesture it is the Hand Gesture Recognition Training Dataset. It contains images of gestures at different angles and clarity which helps our model to predict the hand gesture even in less clarity.

## **Literature Review (all papers considered in a team):**

### **Review on Various Schemes:**

#### **Hand Gesture Recognition Using CNN for Post-Stroke People:**

The main idea was to make a Hand Recognition system for a post-stroke or disabled people by which they can convey their message easily, for that the 3D-CNN was used and a dataset of 7 different hand gesture representing some message which we feel difficult to understand in real life. The dataset consists of hand gesture with less distance from camera and many different types of hands (young people hands, old people's hand etc.)

Citation - Alnaim, Norah, Maysam Abbod, and Abdulrahman Albar. "Hand gesture recognition using convolutional neural network for people who have experienced a stroke." *2019 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies (ISMSIT)*. IEEE, 2019.

#### **Static Hand Gesture Recognition using Convolutional Neural Network with Data**

##### **Augmentation:**

Recent research has confirmed the superiority of Convolutional Neural Network (CNN) for image likeness and categorization. Since, CNN can determine complex and non-linear friendships with figures, in this paper, a changeless help sign recognition system utilizing CNN was projected. Data augmentation like re-measuring, zooming, shearing, turn, breadth and height fluctuating was used to the dataset. They also pre-processed the data dataset to reduce the computational complication and achieve better efficiency.

Citation – Islam, Md Zahirul, et al. "Static hand gesture recognition using convolutional neural network with data augmentation." *2019 Joint 8th International Conference on Informatics, Electronics & Vision (ICIEV) and 2019 3rd International Conference on Imaging, Vision & Pattern Recognition (icIVPR)*. IEEE, 2019.

#### **IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition:**

They used a live cam input and 3D-CNN model to predict the hand gesture and to increase the usage of hand gestures they also introduce hand gestures in the dataset which make more than one actions

to do a task as this can increase the functionality of the system. For example, throw-left, double-click, zoom-in, zoom-out etc. This dataset is named as IPN Hand with sufficient size, variety, and real-world elements able to train and evaluate deep neural networks.

Citation – Benitez-Garcia, Gibran, et al. "IPN hand: A video dataset and benchmark for real-time continuous hand gesture recognition." *2020 25th International Conference on Pattern Recognition (ICPR)*. IEEE, 2021.

### **Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks:**

They secondhand a sliding dormer accompanying a stride that run through arriving broadcast frames where indicator sequence placed at the starting point of classifier queue. If the indicator understands an action/action, then the classifier is mobilized. The indicator's output is post-treated for a more robust acting, and the indispensable content is made utilizing distinct-time incitement block where alone incitement occurs per acted gesture.

Citation – Köpüklü, Okan, et al. "Real-time hand gesture detection and classification using convolutional neural networks." *2019 14th IEEE International Conference on Automatic Face & Gesture Recognition (FG 2019)*. IEEE, 2019.

### **Hand Recognition Using Geometric Classifiers:**

Classification, verification and identification of the input images were done using two simple geometric classifiers. They have also tested system using nearest box classifier and MEB classifier. They were getting good results in MEB classifier and they used geometric classifier because of the fact, this method was designed to work on data that are heavily skewed by a preponderance of duplicate values.

Citation – Bulatov, Yaroslav, et al. "Hand recognition using geometric classifiers." *International Conference on Biometric Authentication*. Springer, Berlin, Heidelberg, 2004.

### **IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition:**

The idea here is to make a hand gesture recognition system that can recognize all the hand gesture signs. By using the dataset, we will be recognizing the hand gesture signs while playing the video from the dataset, and also, we will be making the system efficient to recognize the gesture in

different types of environments which can help us to avoid making a specific environment for the system to be active. Remembering the hand sign of every recognition can be difficult for the user, so we will be using some common signs that users use in their daily life and can easily be remembered.

How it is going to be done:

For Hand gesture recognition we will be using Artificial neural network technique and mainly the Convolution neural network (CNN) and for implementation we will be using python.

The dataset we will be using is IPN Hand.

The dataset contains RGB videos, which are recorded in the resolution of  $640 \times 480$  with a frame rate of 30 fps.

We will be dividing the hand gesture recognition into two stages

- i. Gesture spotting: This aims to detect temporal video segments which contain gesture instances.
- ii. Gesture classification: This aims to classify the gesture of each spotted segment.

Citation - Benitez-Garcia, G., Olivares-Mercado, J., Sanchez-Perez, G., & Yanai, K. (2021, January). IPN hand: A video dataset and benchmark for real-time continuous hand gesture recognition. In *2020 25th International Conference on Pattern Recognition (ICPR)* (pp. 4340-4347). IEEE.

### **Real-time hand tracking and gesture recognition system:**

In this paper, hand gesture recognition system to recognize real time gesture in unconstrained environments. The system consists of three modules: real time hand tracking, training gesture and gesture recognition using pseudo two dimension hidden Markov models (P2-DHMMs). Using a Kalman filter and hand blobs analysis for hand tracking to obtain motion descriptors and hand region. uses skin color for hand gesture tracking and recognition. The basic idea lies in the real-time generation of gesture model for hand gesture recognition in the content analysis of video sequence from CCD camera. Since hand images are two dimensional, it is natural to believe that the 2-DHMM, an extension to the standard HMM, will be helpful and offer a great potential for analyzing and recognizing gesture patterns. The complete system works at about 25 frames/sec. If the hand moves too fast Kalman filter at this will not find hand pixels correctly, the tracker starts going out of track due to the translation and scale parameters get distorted values. Beside this if initialization is not good, the tracker can not archived the result of tracking.

Citation - Binh, Nguyen Dang, Enokida Shuichi, and Toshiaki Ejima. "Real-time hand tracking and

gesture recognition system." *Proc. GVIP* (2005): 19-21.

### **Development of hand gesture recognition system using machine learning. *Journal of Ambient Intelligence and Humanized Computing*:**

Hand gesture segmentation usually involves a pre processing stage that removes the unwanted noise. To remove noise, filters like morphological filter Kalman filter and other common filters like median and Gaussian filters have been used in many researches. The actual process of gesture segmentation involves separating the hand gesture from its background image. Skin colour is a very distinctive characteristic of the human hand which can be used to separate it from the background. The proposed HGR system was trained and tested using Sebastian Marcel static hand posture database (Marcel 2019). The dataset has hand gestures from 10 different persons and on complex as well as plain backgrounds. Different pictures are subjected to different illumination and scale conditions. For each gesture, it was made sure that different background and different rotation of the gesture was taken into consideration.

#### **Algorithm:-**

Hand gesture recognition involves multiple stages of processing so it will be necessary to develop suitable algorithms for each stage of the process. The main stages can be divided into: hand segmentation, feature extraction and classification.

1. The first stage of the HGR system is the image segmentation where the hand will be isolated from its background and then the image will be prepared for its further stages.
2. In this step, the colour distribution is separated into skin colour and non-skin colour and then the threshold value is calculated using the histogram method.
3. Feature extraction methodology
  1. The integral of the image is taken.
  2. Key points are localized using Fast-Hessian.
  3. Orientation assignment is performed.
  4. Descriptors are extracted from the identified key points

#### **Gesture classification**

After successfully extracting the required features and obtaining a fixed size input vector, the classification is implemented using a multiclass support vector machine. The system was tested with a wide range of images which are rotated as well as scaled and it has shown to perform with an overall accuracy of 96.5% with a very fast recognition time of 0.024 s. This clearly shows that the HGR system is rotation and scale invariant and can accept gestures from complex backgrounds. Due



to the fast recognition time, this system can be enhanced by employing real time gesture images and developing the classifier accordingly.

Citation - Parvathy, Priyanka, et al. "Development of hand gesture recognition system using machine learning." *Journal of Ambient Intelligence and Humanized Computing* 12.6 (2021): 6793-6800.

### **Real-time hand tracking and gesture recognition system using neural networks:**

A simple and fast algorithm using orientation histograms is developed. It will recognize a subset of static hand gestures. A pattern recognition system will be using a transform that converts an image into a feature vector, which will be compared with the feature vectors of a training set of gestures. The final system will be Perceptron implementation in MATLAB. This paper includes experiments of 33 hand postures and discusses the results. Experiments shows that the system can achieve a 90% recognition average rate and is suitable for real time applications. The dataset is Various Myanmar Alphabet Language databases on the Internet and photographs that are taken with a digital camera. This meant that they have different sizes, different resolutions and some times almost completely different angles of shooting.

Citation - Maung, Tin Hninn Hninn. "Real-time hand tracking and gesture recognition system using neural networks." *World Academy of Science, Engineering and Technology* 50 (2009): 466-470.

### **An automatic hand gesture recognition system based on Viola-Jones method and SVMs:**

The system consists of two modules: hand gesture detection module and hand gesture recognition module. The detection module could accurately locate the hand regions with a blue rectangle; this is mainly based on Viola-Jones method, which is currently considered the fastest and most accurate learning-based method for object detection. In the recognition module, the Hu invariant moments feature vectors of the detected hand gesture are extracted and a Support Vector Machines (SVMs) classifier is trained for final recognition, due to its high generalization performance without the need to add a priori knowledge. The performance of the proposed system is tested through a series of experiments, the proposed system tries to apply Viola-Jones method to real-time hand gesture detection. The method was mainly realized by three parts

- Using a set of Haar-like features and integral image representation for feature extraction.
- During training, using AdaBoost algorithm to select the critical features among a large number of extracted features stage by stage and finally yields an extremely efficient classifier.

- Combining the strong classifiers in a cascade structure which efficiently increases the speed of the detector by focusing attention on promising regions of the image.

Citation – Yun, Liu, and Zhang Peng. "An automatic hand gesture recognition system based on Viola-Jones method and SVMs." *2009 Second International Workshop on Computer Science and Engineering*. Vol. 2. IEEE, 2009.

### Comparative study on various subtitles:

(Title, Year, Authors)	Methodology or Techniques used (Mention specific algorithms or recent technologies)	Advantages	Issues	Metrics used (those are used to justify the performance of the used scheme)
Hand Gesture Recognition Using CNN for Post-Stroke People, 2019, Norah Alnaim, Abdulrahman Albar, and Maysam F Abbod	CNN	The accuracy result of training is 100% compared to that of testing and specificity in training is 100% whereas in testing is 99.89%. Its execution time is approximate 15,598 seconds, which is duration to train and test the system using seven hand gestures which are used in the experiment. Overall, training has the best values in most parameters.	It fails to identify hand gesture if the distance between hand and camera is more.	execution time, accuracy, sensitivity, specificity, positive and negative predictive value, likelihood and root mean square.
Static Hand Gesture Recognition using Convolutional Neural Network with Data Augmentation, 2019, Md. Zahirul Islam, Mohammad	CNN	The model with augmented data achieved accuracy 97.12% which is nearly 4% higher than the model without augmentation (92.87%).	Recognition of gestures made with both hands is not possible by this system.	Loss, and Accuracy

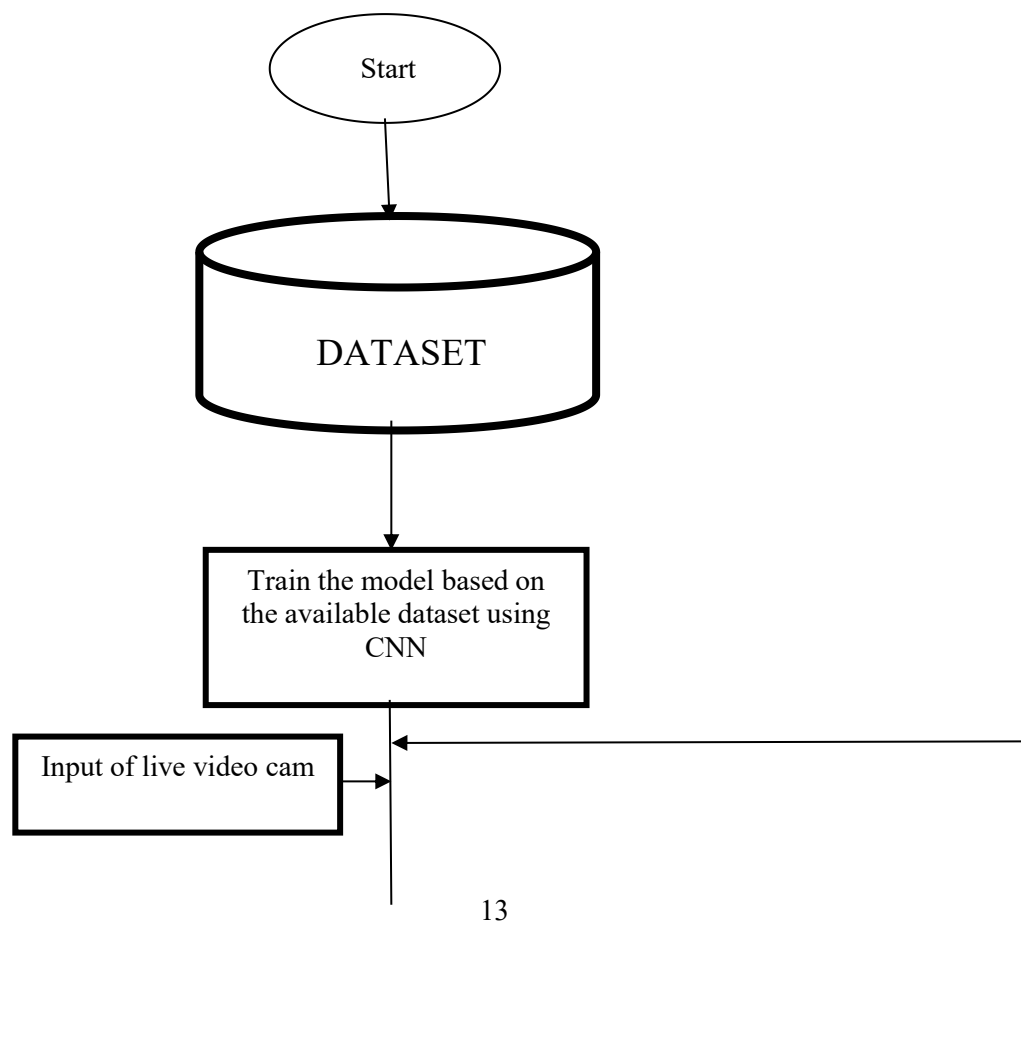
Shahadat Hossain, Raihan Ul Islam, and Karl Andersson				
IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition, 2020, Gibran Benitez-Garcia, Jesus Olivares-Mercado , Gabriel Sanchez-Perez, and Keiji Yanai	CNN	It can group the frame (i.e., more than one action) and find which notation it makes. More commonly called as complete animation instead of a picture.	The model sizes are much bigger and it requires more processor for predicting the gesture.	conventional classification, Levenshtein accuracy, and binary classification accuracy
Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks, 2019, Okan Kopuklu, Ahmet Gunduz , Neslihan Kose , and Gerhard Rigoll	CNN	We obtain considerable early detections while achieving performances close to offline operation.	It can only predict whether the gesture is present in the image or not, nothing else.	Levenshtein distance
Hand Recognition Using Geometric Classifiers, 2004, Yaroslav Bulatov, Sachin	Geometric Classifiers	We describe a novel minimum enclosing ball classifier which performs well for hand recognition and could be of interest for other applications. Our system is easier to	It can only use for medium security applications.	$L_\infty$ metric

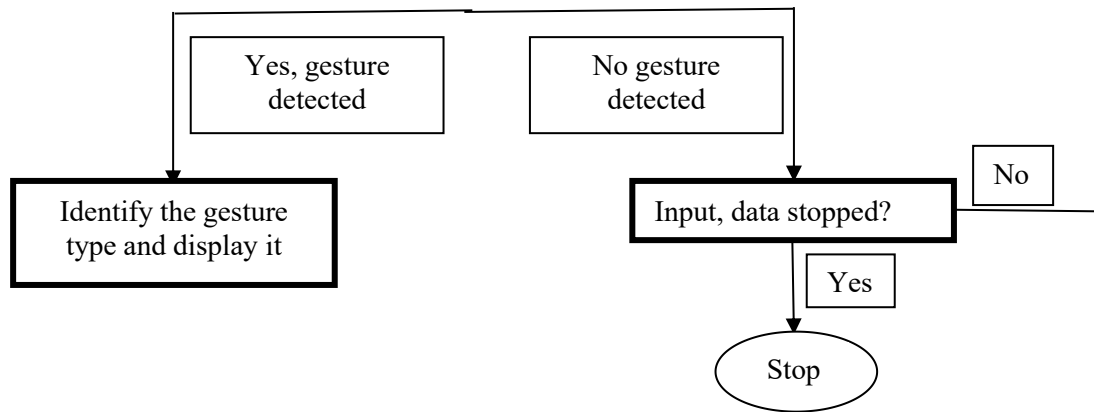
Jambawalikar, Piyush Kumar, and Saurabh Sethia1		use, cheaper to build and more accurate than previous systems.		
IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition, 2020, Gibran Benitez-Garcia, Jesus Olivares-Mercado , Gabriel Sanchez-Perez, and Keiji Yanai	Convolutional Neural Network	It can group the frame (i.e., more than one action) and find which notation it makes. More commonly called as complete animation instead of a picture.	The model sizes are much bigger and it requires more processor for predicting the gesture.	Classification Confusion Matrix of Prediction Metric and conventional classification-Levenshtein accuracy, and binary classification accuracy
Real-Time Hand Tracking and Gesture Recognition System-2009 Nguyen Dang Binh, Enokida Shuichi	Kalman filter and hand blobs analysis for hand tracking	The users do not need to wear a glove, neither is there need for a uniform background. Experiments on a single hand database have been carried out and recognition accuracy of up to 98% has been achieved.	Currently, tracking method is limited to 2D.  If the hand moves too fast Kalman filter at this will not find hand pixels correctly, the tracker starts going out of track	Classification Accuracy metric
Development of hand gesture recognition system using machine learning(2020 )- Priyanka Parvathy Kamalraj Subramaniam	Using median or Gaussian filters. Histogram method for skin-color detection Bag of features method for feature extraction and classification	HGR system is rotation and scale invariant and can accept gestures from complex backgrounds. Has fast recognition time	‘Stop’ gesture showed maximum misclassification. This is because of the similarity of the Stop gesture with the Spread; many gestures have been misclassified between these two classes.	Classification ROC curve Metric

"Real-time hand tracking and gesture recognition system using neural networks."- 2009 Maung, Tin Hninn Hninn	Hidden Markow methods and orientation Histogram for gesture Recognition.	Another advantage of using neural networks is that you can draw conclusions from the network output. If a vector is not classified we can check its output and work out a solution	Implemented with MATLAB which tends to slow down with loops so sometimes back-tracing was avoided in code	Classification Accuracy metric
"An automatic hand gesture recognition system based on Viola-Jones method and SVMs."- Yun, Liu, and Zhang Peng.	Viola Jones Method, Haar like features sets, Adaptive boosting algorithm	It uses SVMs which always find a global minimum because it usually tries to minimize a bound on the structural risk, rather than the empirical risk.	the detection speed for the classifiers using the extended Haar-like feature is somewhat slow down due to the extra computation,	Classification Accuracy Metric

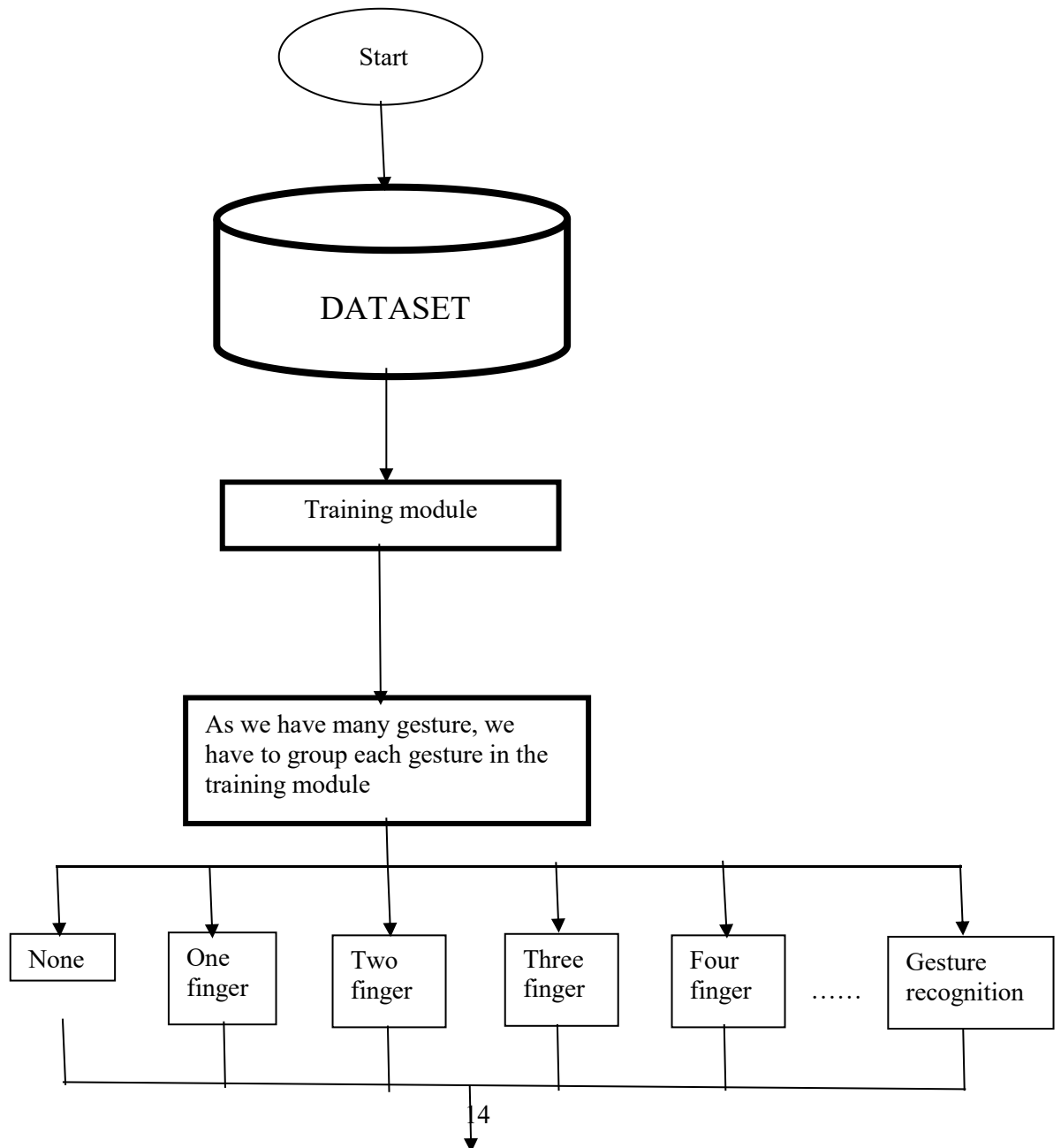
## System design:

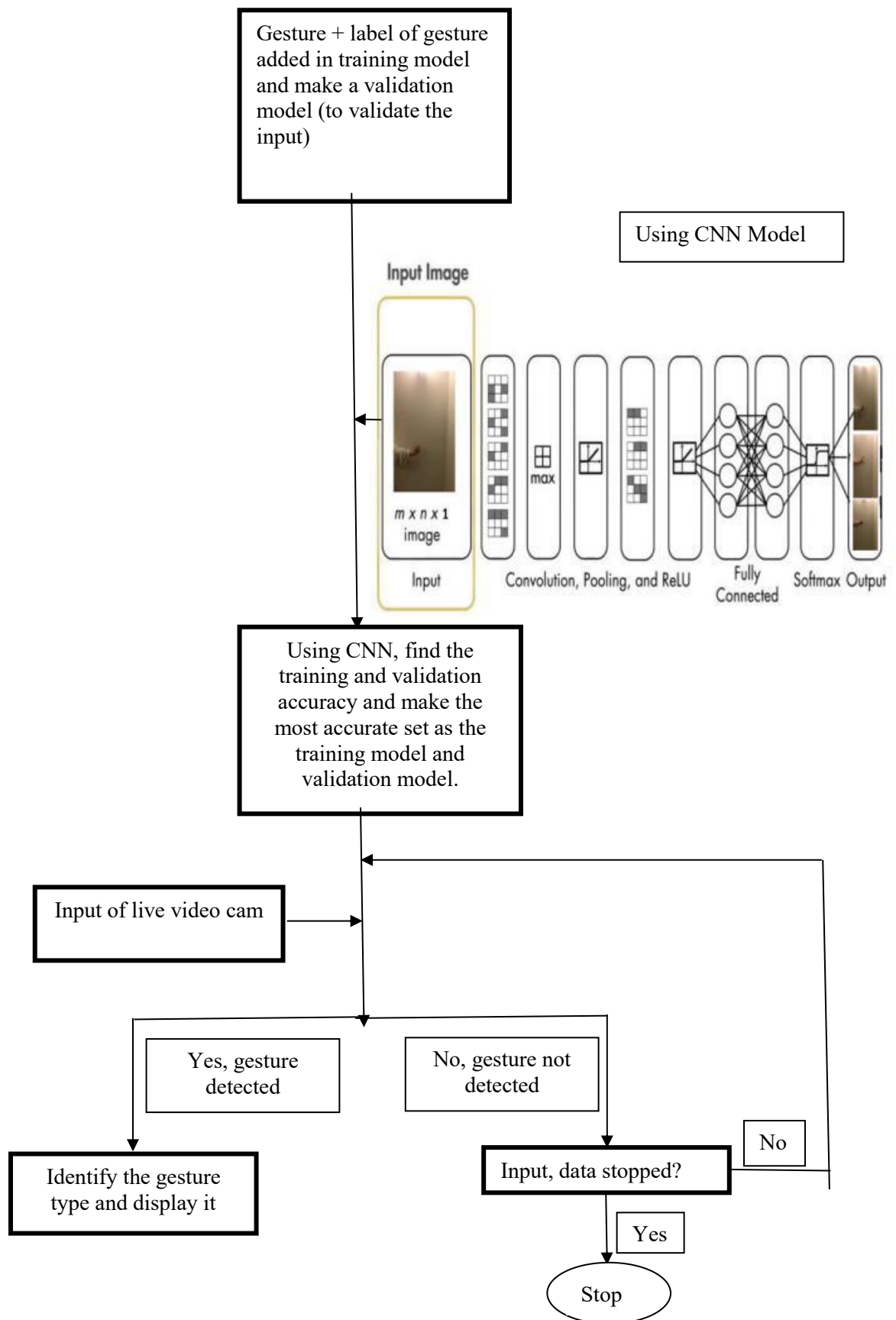
### Architecture Diagram / Flow Diagram / Flowchart / ...





## Detailed Description of Modules





## Software Requirement Specifications:

Soft Computing technique: Convolutional Neural Network (CNN)

Implementation tool:

Operating system: Windows

Coding platform: Pycharm (IDLE Python)

Modules: TensorFlow, OpenCV, Keras, and NumPy.

## Experimental Results & Discussion:

### Source code:

**training.py:**

# Check the accuracy

```
import os
import math
import matplotlib.image
import tensorflow as tf
from tensorflow import keras
from keras.models import Sequential
from keras.layers import Dense, Flatten, Conv2D, MaxPool2D
from keras.optimizers import Adam, SGD
from keras.preprocessing.image import ImageDataGenerator
import matplotlib.pyplot as plt
import warnings
import numpy as np
import cv2
from keras.callbacks import ReduceLROnPlateau
from keras.callbacks import ModelCheckpoint, EarlyStopping
import json
```

```
warnings.simplefilter(action='ignore', category=FutureWarning)
```

```
file = open("checkpoints.json")
data = json.load(file)
```

```
train_path = data['paths'][0]['path_train_data']
test_path = data['paths'][0]['path_test_data']
```

```
train_batches = ImageDataGenerator(
```

```
    preprocessing_function=tf.keras.applications.vgg16.preprocess_input).flow_from_directory(directory
=train_path,
```

```
    target_size=(64, 64),
    class_mode='categorical',
```



```

batch_size=10,
shuffle=True)

test_batches = ImageDataGenerator(

preprocessing_function=tf.keras.applications.vgg16.preprocess_input).flow_from_directory(directory
=test_path,

target_size=(64, 64),
class_mode='categorical',
batch_size=10,
shuffle=True)

imgs, labels = next(train_batches)
labels = os.listdir(train_path)

# Plotting the images...
def plotImages(images_arr):
    fig = plt.figure(figsize=(10, 7))
    rows = math.ceil(len(labels) / 2)
    cols = math.ceil(len(labels) / 5)

    images = []
    folders = os.listdir(train_path)
    for i in folders:
        files = os.listdir(os.path.join(train_path, i))
        paths = [os.path.join(train_path, os.path.join(i, filename)) for filename in files]
        images.append(max(paths, key=os.path.getctime))

    for i in images:
        img = matplotlib.image.imread(i)
        img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
        fig.add_subplot(rows, cols, images.index(i) + 1)
        plt.imshow(img)
        plt.axis('off')
        plt.title(labels[images.index(i)])

    plt.subplots_adjust(hspace=0.5, wspace=0)
    plt.savefig("Result.png", bbox_inches='tight')
    plt.show()

plotImages(imgs)
print(imgs.shape)
print(labels)

word_dict = {}

```

```

for i in range(0, len(labels)):
    word_dict[i] = labels[i]

model = Sequential()

model.add(Conv2D(filters=32, kernel_size=(3, 3), activation='relu', input_shape=(64, 64, 3)))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=64, kernel_size=(3, 3), activation='relu', padding='same'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Conv2D(filters=128, kernel_size=(3, 3), activation='relu', padding='valid'))
model.add(MaxPool2D(pool_size=(2, 2), strides=2))

model.add(Flatten())

model.add(Dense(64, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(128, activation="relu"))
model.add(Dense(len(labels), activation="softmax"))

model.compile(optimizer=Adam(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0001)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0, mode='auto')

model.compile(optimizer=SGD(learning_rate=0.001), loss='categorical_crossentropy',
metrics=['accuracy'])
reduce_lr = ReduceLROnPlateau(monitor='val_loss', factor=0.2, patience=1, min_lr=0.0005)
early_stop = EarlyStopping(monitor='val_loss', min_delta=0, patience=2, verbose=0, mode='auto')

history2 = model.fit(train_batches, epochs=10, callbacks=[reduce_lr, early_stop],
                    validation_data=test_batches) # , checkpoint])
imgs, labels = next(train_batches) # For getting next batch of imgs...

imgs, labels = next(test_batches) # For getting next batch of imgs...
scores = model.evaluate(imgs, labels, verbose=0)
print(model.metrics_names, scores)
print(f'{model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1] * 100}%')

model.save('model.h5')

print(history2.history)

imgs, labels = next(test_batches)

```

```

model = keras.models.load_model(r"model.h5")

scores = model.evaluate(imgs, labels, verbose=0)
print(f {model.metrics_names[0]} of {scores[0]}; {model.metrics_names[1]} of {scores[1] * 100}%)

scores # [loss, accuracy] on test data...
model.metrics_names

predictions = model.predict(imgs, verbose=0)
print("predictions on a small set of test data--")
print("")
for ind, i in enumerate(predictions):
    print(word_dict[np.argmax(i)], end=' ')

# plotImages(imgs)
print('Actual labels')
for i in labels:
    print(word_dict[np.argmax(i)], end=' ')

print(imgs.shape)

# Model accuracy graph
plt.plot(history2.history['accuracy'])
plt.plot(history2.history['val_accuracy'])
plt.title('model accuracy')
plt.ylabel('accuracy')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

# Model loss graph
plt.plot(history2.history['loss'])
plt.plot(history2.history['val_loss'])
plt.title('model loss')
plt.ylabel('loss')
plt.xlabel('epoch')
plt.legend(['train', 'val'], loc='upper left')
plt.show()

history2.history

main.py:
import os
import numpy as np
import mediapipe as mp
import cv2

```

```

import keras
import json

file = open('checkpoints.json')
data = json.load(file)

model = keras.models.load_model(data['paths'][0]['path_model'])

labels = os.listdir(data['paths'][0]['path_train_data'])

background = None
accumulated_weight = 0.5

ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350

word_dict = {}

for i in range(0, len(labels)):
    word_dict[i] = labels[i]

def cal_accum_avg(frame, accumulated_weight):
    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)

def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)

    _, thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY_INV)

    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        return (thresholded)
    else:

```

```

        hand_segment_max_cont = max(contours, key=cv2.contourArea)
        return (thresholded, hand_segment_max_cont)

cam = cv2.VideoCapture(0, cv2.CAP_DSHOW)
num_frames = 0

while True:
    ret, frame = cam.read()

    # flipping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(frame, 1)

    frame_copy = frame.copy()

    # ROI from the frame
    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)

    if num_frames < 70:
        cal_accum_avg(gray_frame, accumulated_weight)

        cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT", (80, 400),
            cv2.FONT_HERSHEY_SIMPLEX, 0.9,
            (0, 0, 255), 2)

    else:
        # segmenting the hand region
        hand = segment_hand(gray_frame)

        if len(hand) == 2:
            thresholded, hand_segment = hand

            # Drawing contours around hand segment
            cv2.drawContours(frame_copy, [hand_segment + (ROI_right, ROI_top)], -1, (255, 0, 0), 1)

        else:
            thresholded = hand

    cv2.imshow("Thesholded Hand Image", thresholded)

    thresholded = cv2.resize(thresholded, (64, 64))
    thresholded = cv2.cvtColor(thresholded, cv2.COLOR_GRAY2RGB)
    thresholded = np.reshape(thresholded, (1, thresholded.shape[0], thresholded.shape[1], 3))

```

```

    pred = model.predict(thresholded)
    cv2.putText(frame_copy, word_dict[np.argmax(pred)], (170, 45),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    # Draw ROI on frame_copy
    cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right, ROI_bottom), (255, 128, 0), 3)

    # incrementing the number of frames for tracking
    num_frames += 1

    # Display the frame with segmented hand
    cv2.putText(frame_copy, "DataFlair hand sign recognition __", (10, 20), cv2.FONT_ITALIC,
0.5, (51, 255, 51), 1)
    cv2.imshow("Sign Detection", frame_copy)

    # Close windows with Esc
    k = cv2.waitKey(1) & 0xFF

    if k == 27:
        break

# Release the camera and destroy all the windows
cam.release()
cv2.destroyAllWindows()

```

#### **create\_gesture\_data.py:**

```

import os
import cv2
import glob
import json
import numpy as np
import tensorflow as tf

background = None
accumulated_weight = 0.5

ROI_top = 100
ROI_bottom = 300
ROI_right = 150
ROI_left = 350

file = open('checkpoints.json')
data = json.load(file)

test_path = data['paths'][0]['path_test_data']
path = data['paths'][0]['path_train_data']

```

```

traindata = glob.glob(path + "\*")
noOfGestures = len(traindata)
print("Number of Gestures already available: ", noOfGestures)

def cal_accum_avg(frame, accumulated_weight):
    global background

    if background is None:
        background = frame.copy().astype("float")
        return None

    cv2.accumulateWeighted(frame, background, accumulated_weight)

def segment_hand(frame, threshold=25):
    global background

    diff = cv2.absdiff(background.astype("uint8"), frame)

    _, thresholded = cv2.threshold(diff, threshold, 255, cv2.THRESH_BINARY)

    # Grab the external contours for the image
    contours, hierarchy = cv2.findContours(thresholded.copy(), cv2.RETR_EXTERNAL,
cv2.CHAIN_APPROX_SIMPLE)

    if len(contours) == 0:
        return None
    else:
        hand_segment_max_cont = max(contours, key=cv2.contourArea)
        return thresholded, hand_segment_max_cont

cam = cv2.VideoCapture(0)

num_frames = 0
element = 10
num_imgs_taken = 0
st = ""
res = []

def get_gesture_name(path):
    n = input("Enter the Gesture name: ")
    path = path + "\\" + n
    if not os.path.exists(path):
        os.mkdir(path)
    if not os.path.exists(os.path.join(test_path, n)):

```

```

        os.mkdir(os.path.join(test_path, n))
    return n

name = get_gesture_name(path)

while True:
    ret, frame = cam.read()

    # flipping the frame to prevent inverted image of captured frame...
    frame = cv2.flip(frame, 1)

    frame_copy = frame.copy()

    roi = frame[ROI_top:ROI_bottom, ROI_right:ROI_left]

    gray_frame = cv2.cvtColor(roi, cv2.COLOR_BGR2GRAY)
    gray_frame = cv2.GaussianBlur(gray_frame, (9, 9), 0)

    if num_frames < 60:
        cal_accum_avg(gray_frame, accumulated_weight)
        if num_frames <= 59:
            cv2.putText(frame_copy, "FETCHING BACKGROUND...PLEASE WAIT", (80, 400),
cv2.FONT_HERSHEY_SIMPLEX, 0.9,
                        (0, 0, 255), 2)
            # cv2.imshow("Sign Detection", frame_copy)

    # Time to configure the hand specifically into the ROI...
    elif num_frames <= 299:

        hand = segment_hand(gray_frame)

        cv2.putText(frame_copy, "Adjust hand...Gesture for" + str(element), (200, 400),
cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (0, 0, 255), 2)

    # Checking if hand is actually detected by counting number of contours detected...
    if hand is not None:
        thresholded, hand_segment = hand

        # Draw contours around hand segment
        cv2.drawContours(frame_copy, [hand_segment + (ROI_right, ROI_top)], -1, (255, 0, 0), 1)

        # Also display the thresholded image
        cv2.imshow("Thresholded Hand Image", thresholded)

    else:

```



```

# Segmenting the hand region...
hand = segment_hand(gray_frame)

# Checking if we are able to detect the hand...
if hand is not None:

    # unpack the thresholded img and the max_contour...
    thresholded, hand_segment = hand

    # Drawing contours around hand segment
    cv2.drawContours(frame_copy, [hand_segment + (ROI_right, ROI_top)], -1, (255, 0, 0), 1)

    cv2.putText(frame_copy, str(num_frames), (70, 45), cv2.FONT_HERSHEY_SIMPLEX, 1, (0,
0, 255), 2)
    # cv2.putText(frame_copy, str(num_frames)+"For" + str(element), (70, 45),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0,0,255), 2)
    cv2.putText(frame_copy, str(num_imgs_taken + 1) + 'images For' + str(element), (200, 400),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

    # Displaying the thresholded image
    cv2.imshow("Thresholded Hand Image", thresholded)
    if num_imgs_taken <= 299:
        cv2.imwrite(path + "\\" + name + "\\gest" + str(noOfGestures) + "_" + str(num_imgs_taken
+ 1) + '.jpg',
                    thresholded)
    else:
        break
    num_imgs_taken += 1
else:
    cv2.putText(frame_copy, 'No hand detected...', (200, 400),
cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 2)

# Drawing ROI on frame copy
cv2.rectangle(frame_copy, (ROI_left, ROI_top), (ROI_right, ROI_bottom), (255, 128, 0), 3)

cv2.putText(frame_copy, "DataFlair hand sign recognition___", (10, 20), cv2.FONT_ITALIC,
0.5, (51, 255, 51), 1)

# increment the number of frames for tracking
num_frames += 1

# Display the frame with segmented hand
cv2.imshow("Sign Detection", frame_copy)

# Closing windows with Esc key...(any other key with ord can be used too.)
k = cv2.waitKey(1) & 0xFF

if k == 27:

```

```

        break

# Releasing camera & destroying all the windows...

cv2.destroyAllWindows()
cam.release()

checkpoint.json:
{
  "paths": [
    {
      "path_train_data": "E:/PyCharm/PycharmProjects/SC-Project/Different-Hand-Gesture-Recognition-Using-CNN/Traindata",
      "path_test_data": "E:/PyCharm/PycharmProjects/SC-Project/Different-Hand-Gesture-Recognition-Using-CNN/Testdata",
      "path_model": "E:/PyCharm/PycharmProjects/SC-Project/Different-Hand-Gesture-Recognition-Using-CNN/model.h5",
      "path_dir": "E:/PyCharm/PycharmProjects/SC-Project/Different-Hand-Gesture-Recognition-Using-CNN"
    }
  ]
}

```

**Hand Gesture available in Dataset:**

Awesome



Hello



One finger



Three finger



Two finger



Four finger



None



Right



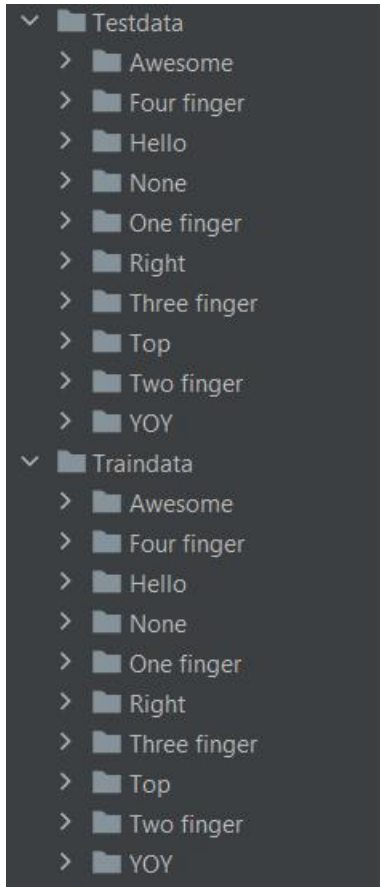
Top



YOY



**Traindata and Testdata folders:**

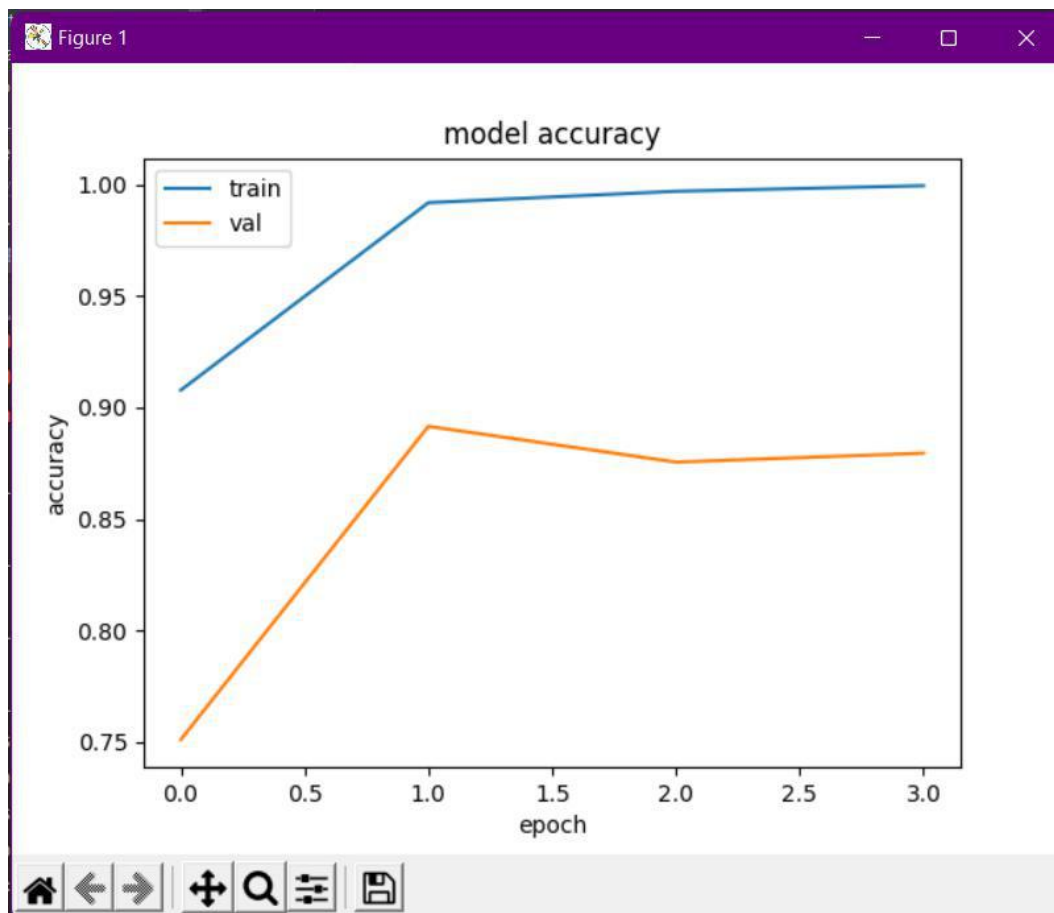


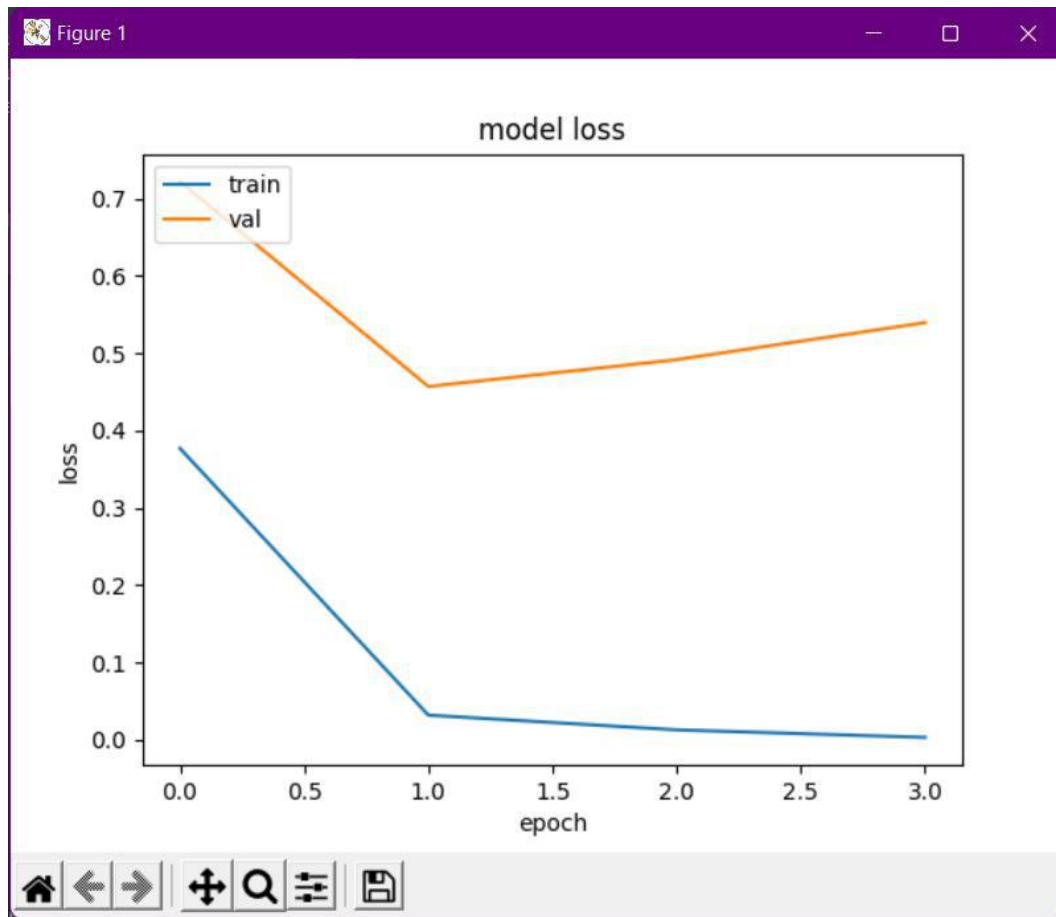
## Screenshots with Explanation:

We have run the training.py file which train our dataset and we have made the estimation that from the best initial epochs, once the accuracy is reached approximately to 100 then we will just stop the training by which we can save some time and the model can be made faster and the loss was also made lesser for the model.

```
Epoch 1/10
1000/1000 [=====] - 66s 65ms/step - loss: 0.3771 - accuracy: 0.9078 - val_loss: 0.7221 - val_accuracy: 0.7510 - lr: 0.0010
Epoch 2/10
1000/1000 [=====] - 25s 25ms/step - loss: 0.0321 - accuracy: 0.9919 - val_loss: 0.4571 - val_accuracy: 0.8916 - lr: 0.0010
Epoch 3/10
1000/1000 [=====] - 29s 29ms/step - loss: 0.0130 - accuracy: 0.9970 - val_loss: 0.4918 - val_accuracy: 0.8755 - lr: 0.0010
Epoch 4/10
1000/1000 [=====] - 28s 28ms/step - loss: 0.0033 - accuracy: 0.9995 - val_loss: 0.5397 - val_accuracy: 0.8795 - lr: 5.0000e-04
['loss', 'accuracy'] [2.1748526096343994, 0.699999988079071]
loss of 2.1748526096343994; accuracy of 69.9999988879871%
{'loss': [0.3770516514778137, 0.032107558101415634, 0.013048245571553707, 0.0033146110363304615], 'accuracy': [0.907800018787384, 0.9919000267982483, 0.996999979019165, 0.9994999]
loss of 0.011292912065982819; accuracy of 100.0%
```

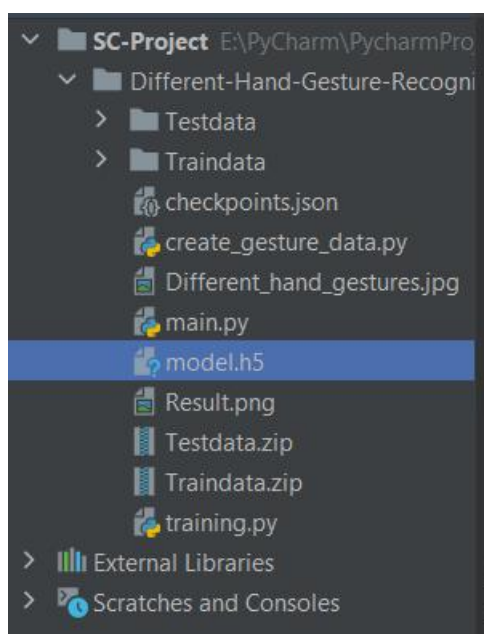
The graphs plotted for accuracy and loss form the epochs are:





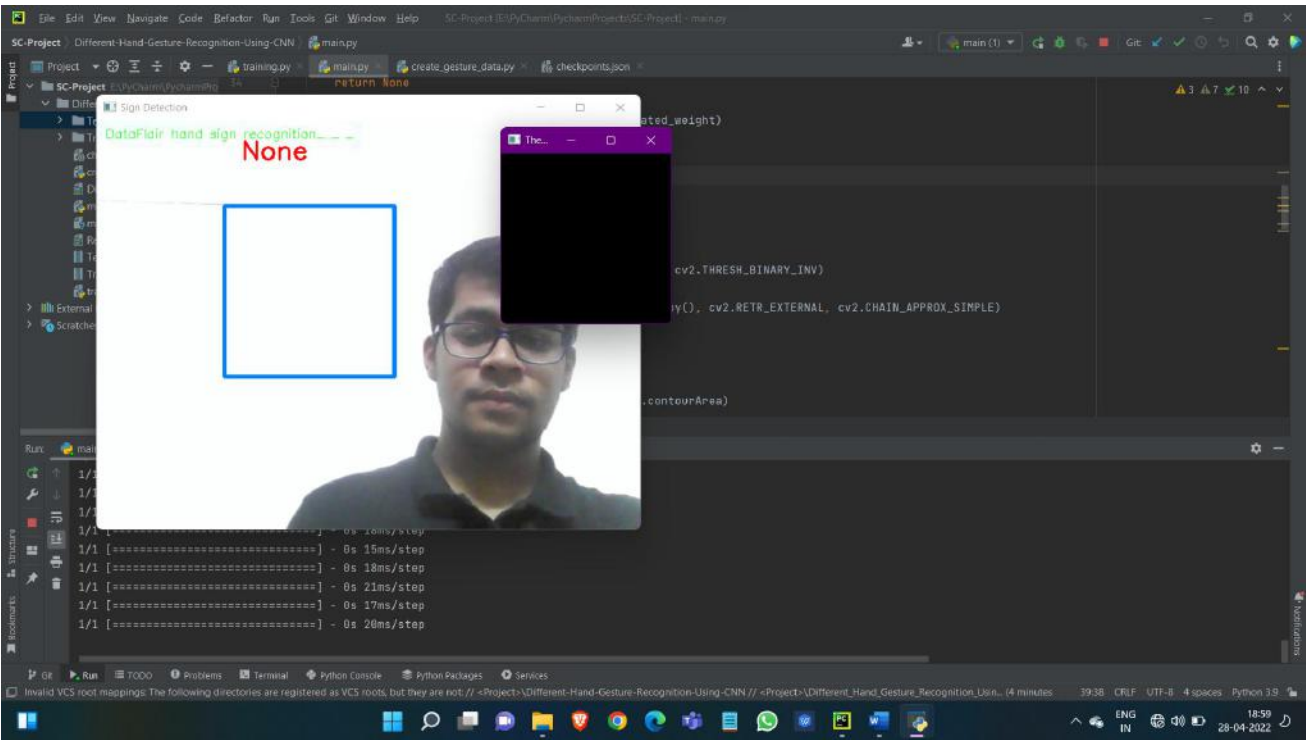
So, the graph represents the accuracy and loss for the training and testing of the model.

We save the training model as .h5 file such that we can directly use it to predict the gesture in the main.py instead of training it again in the main.py file

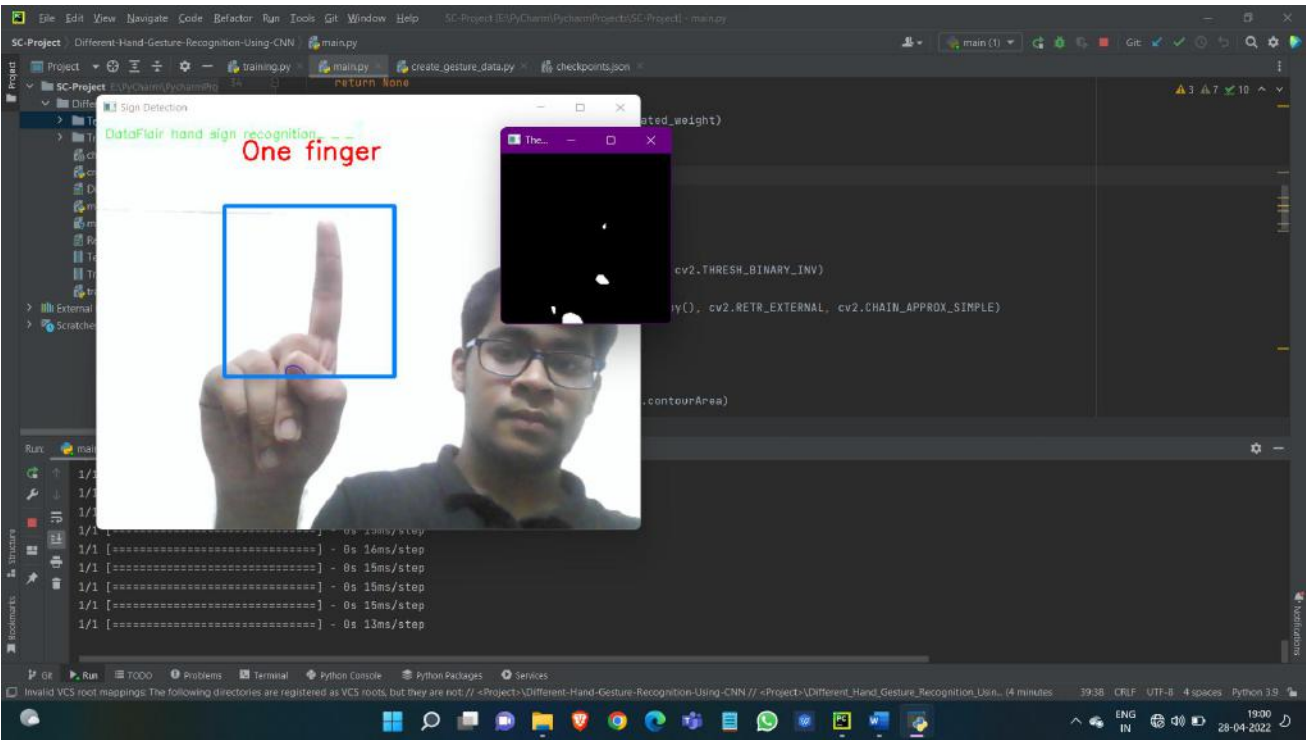


The predictions screenshot from the model are:

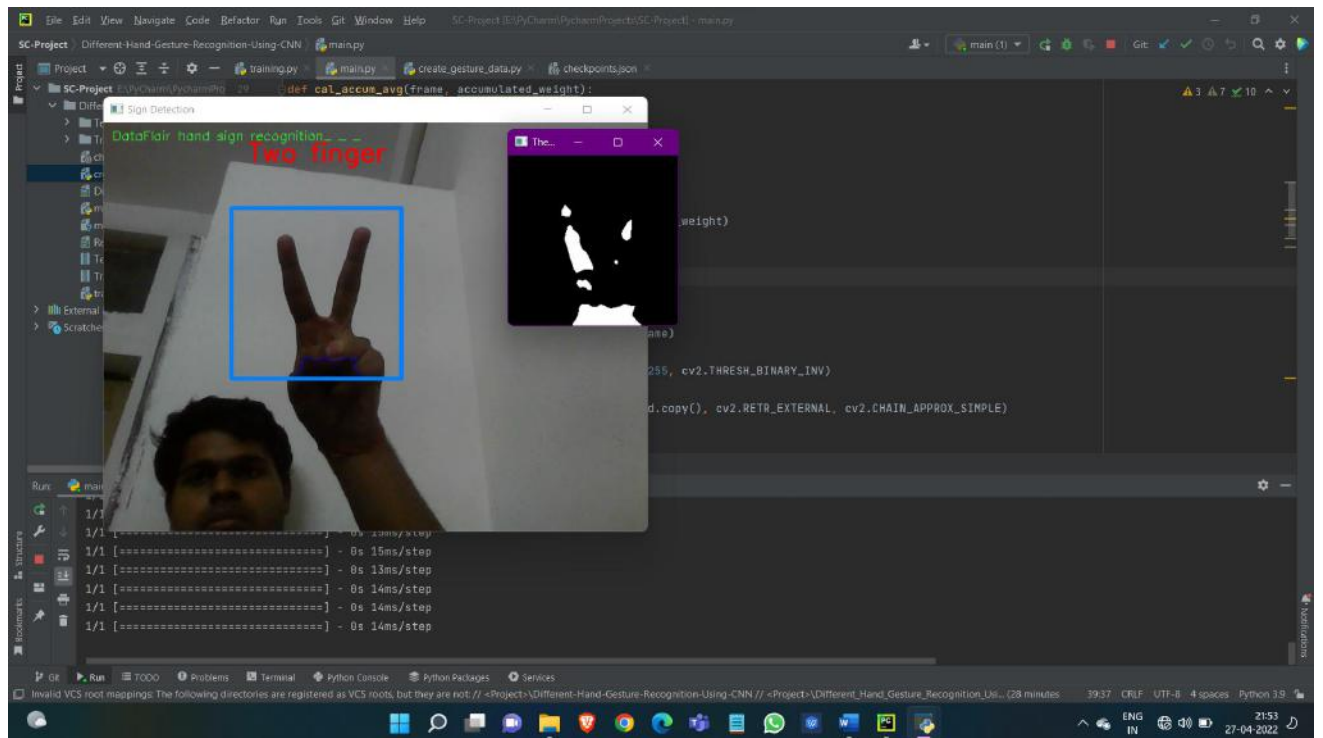
None:



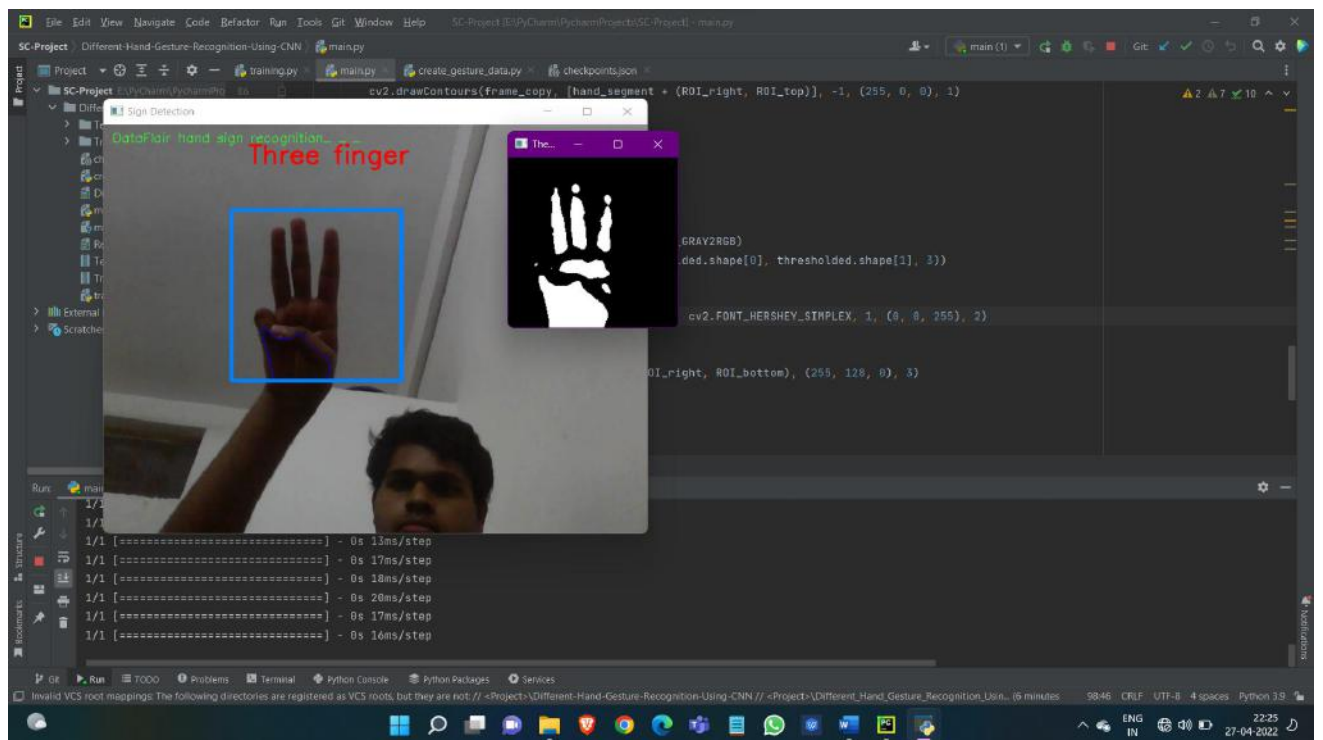
One finger:



Two finger:

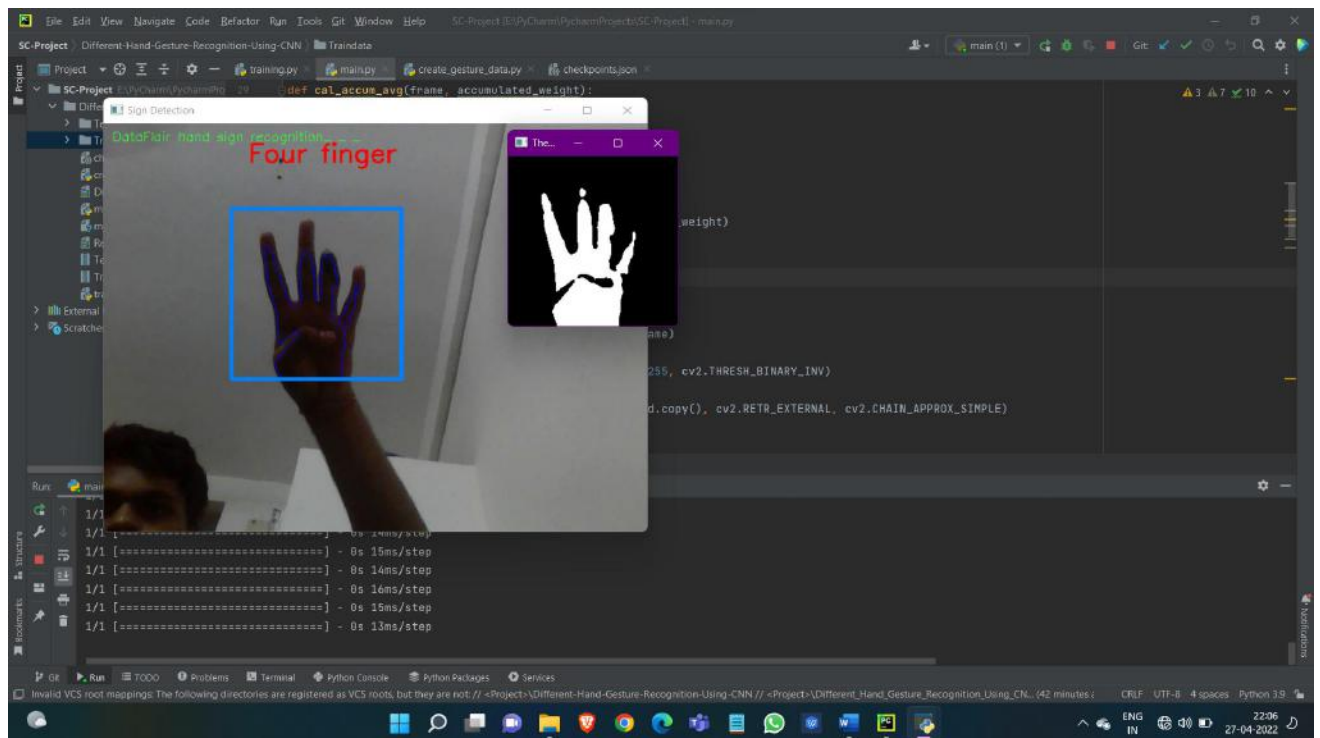


Three finger:

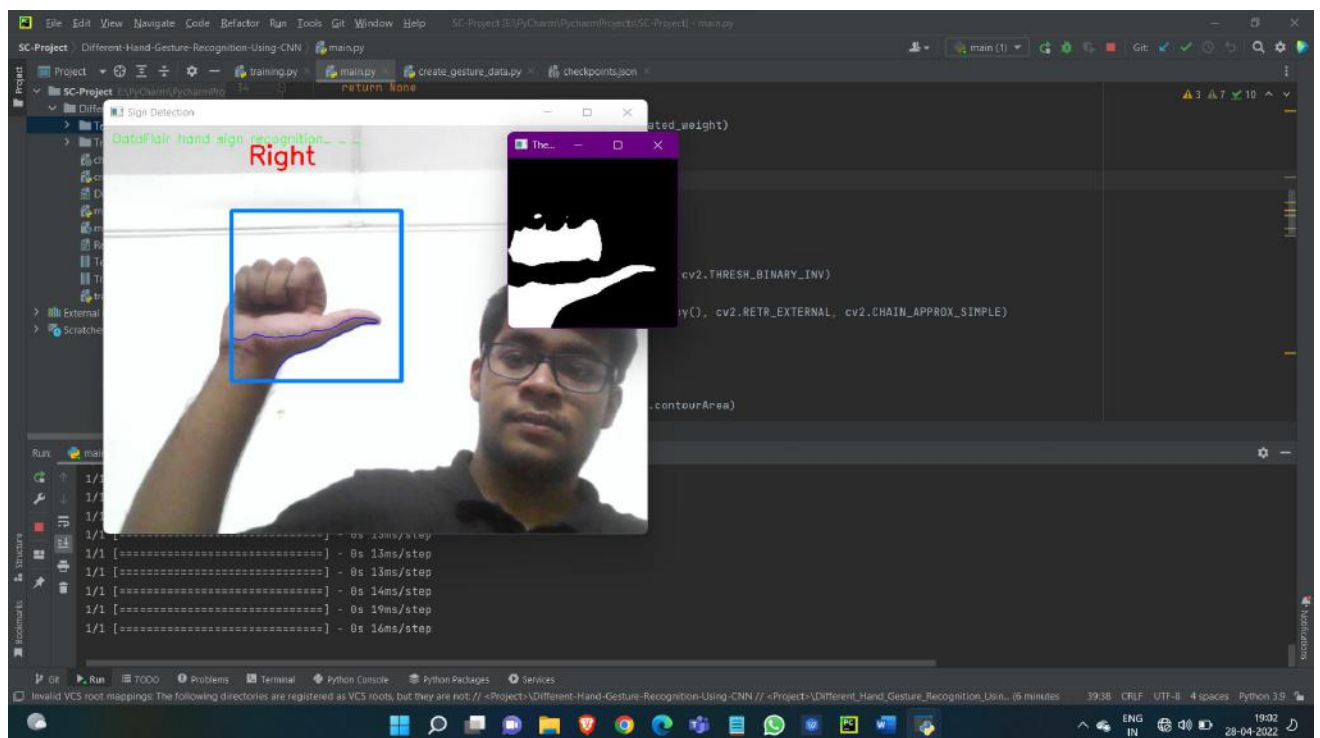




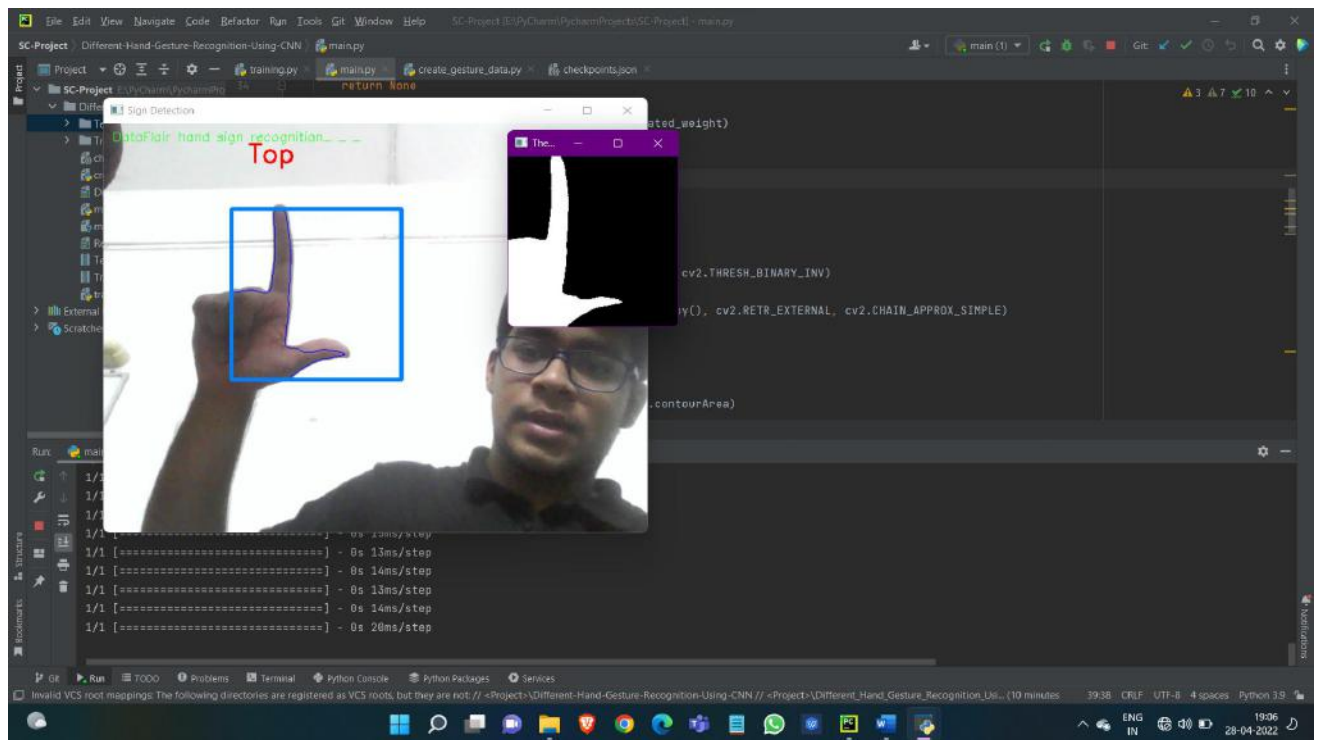
Four  
finger:



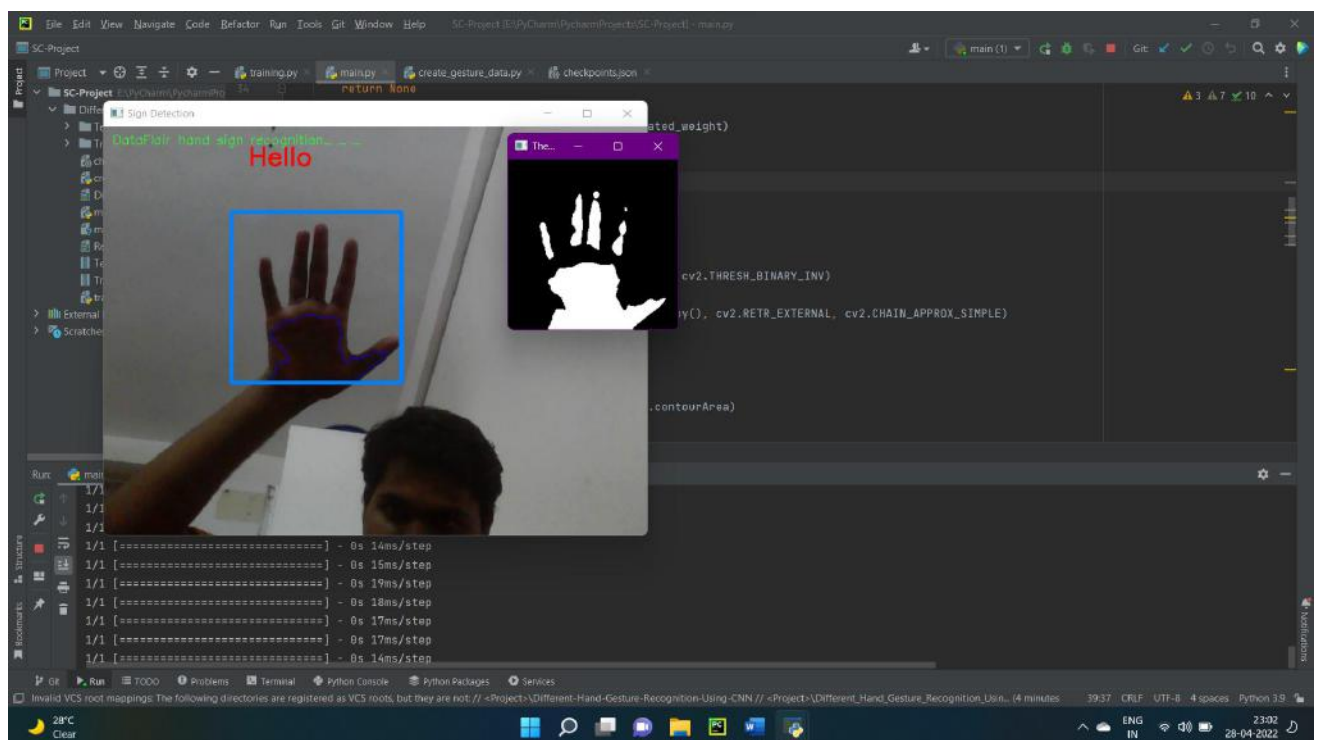
Right:



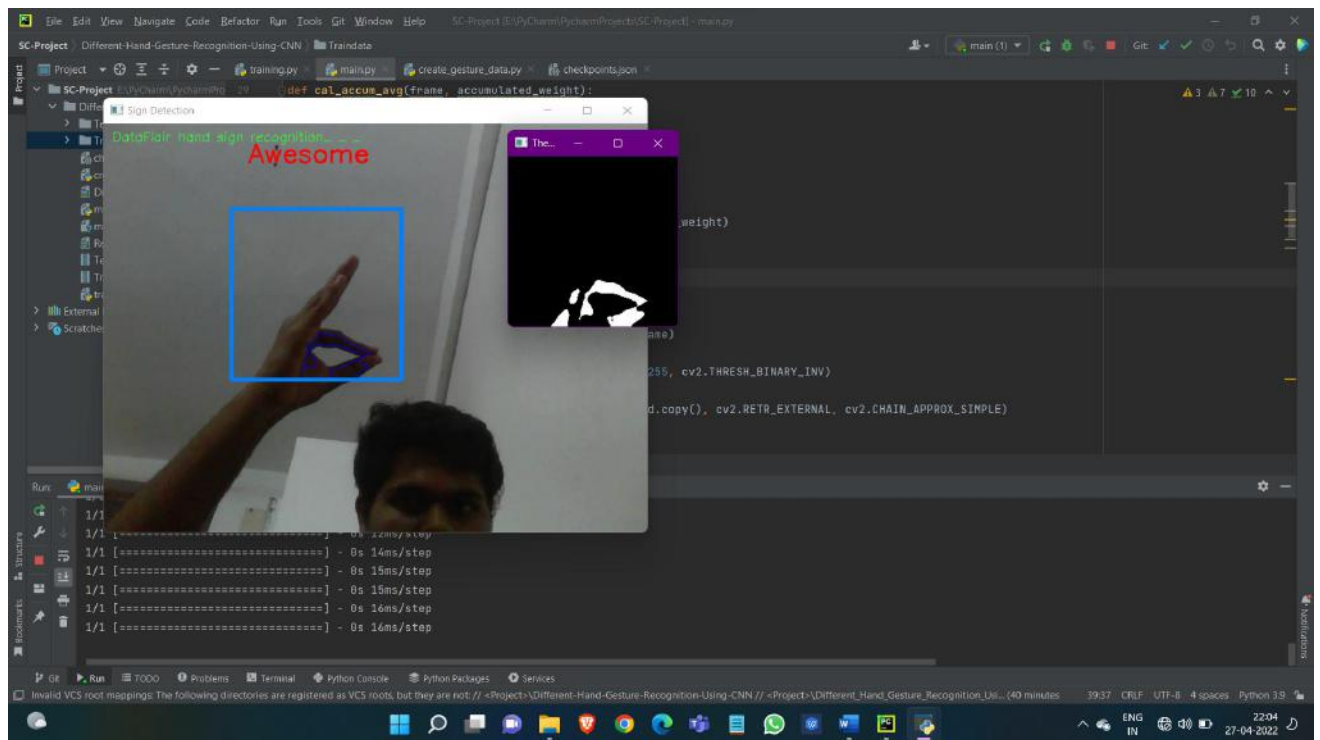
TOP:



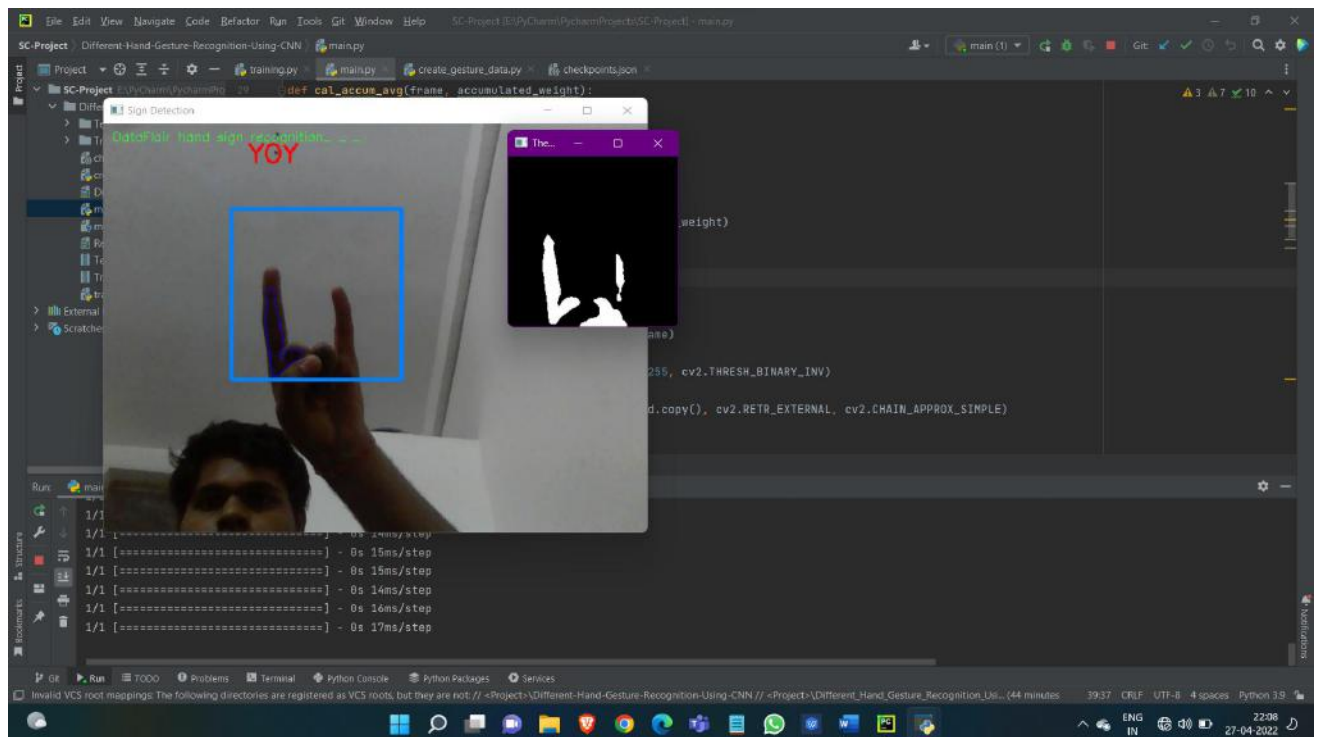
Hello:



Awesome:

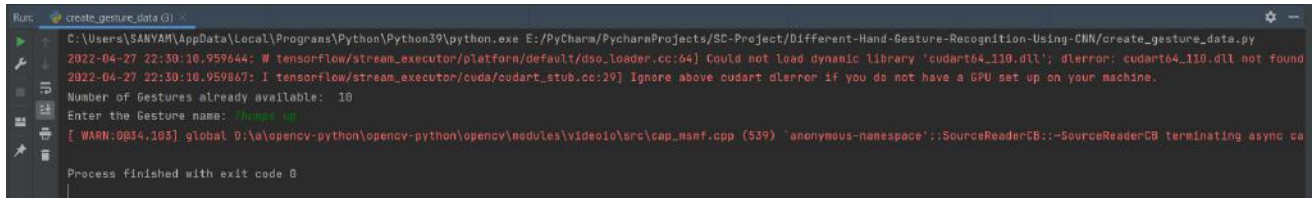


YOY:

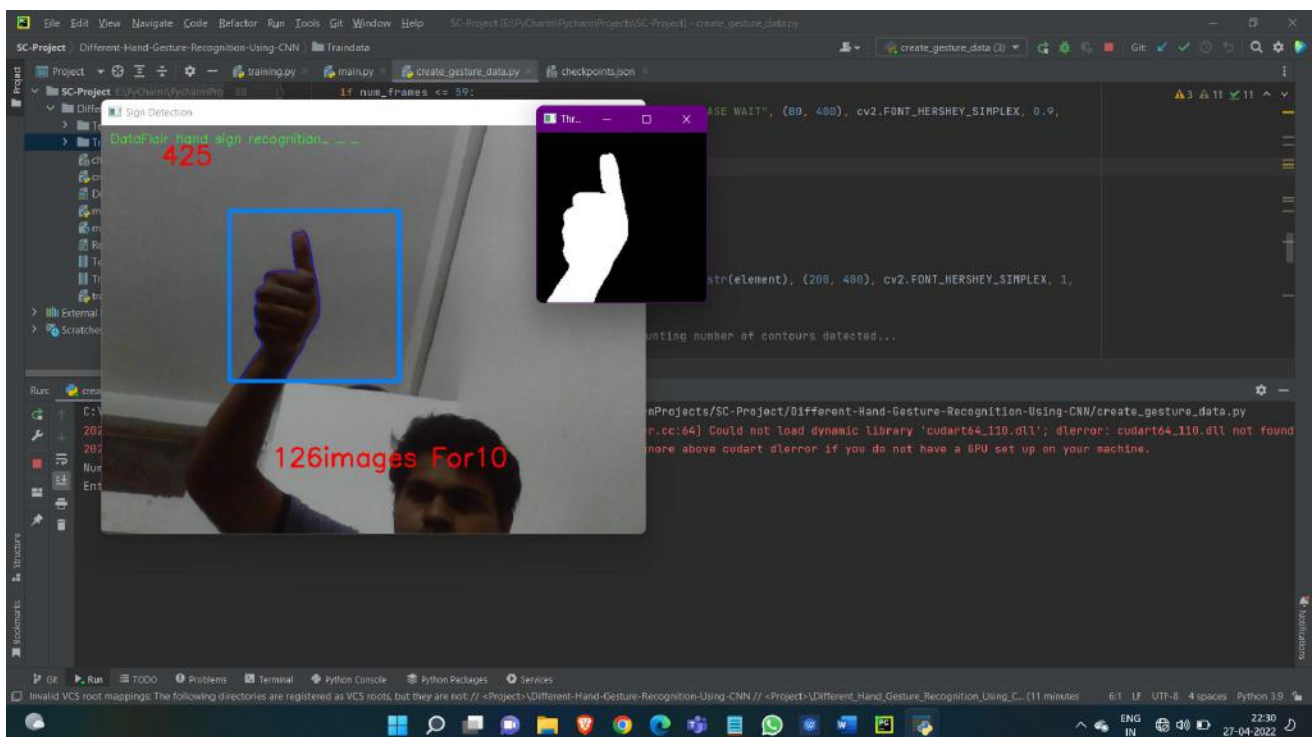


Due to background disturbances the threshold image for the hand gesture (The one in right window) was not clear but our model was able to predict the correct sign based on the training.

In the `create_gesture_data.py` we are asking the user to add a data in the dataset for that we will open a video cam, firstly the user has to enter the name of gesture (sign) such that specific folder for that can be create in the dataset and then user has to show the sign which he want to add in that folder for some time by which we can get approximately 300 images for that gesture and store it in the dataset.

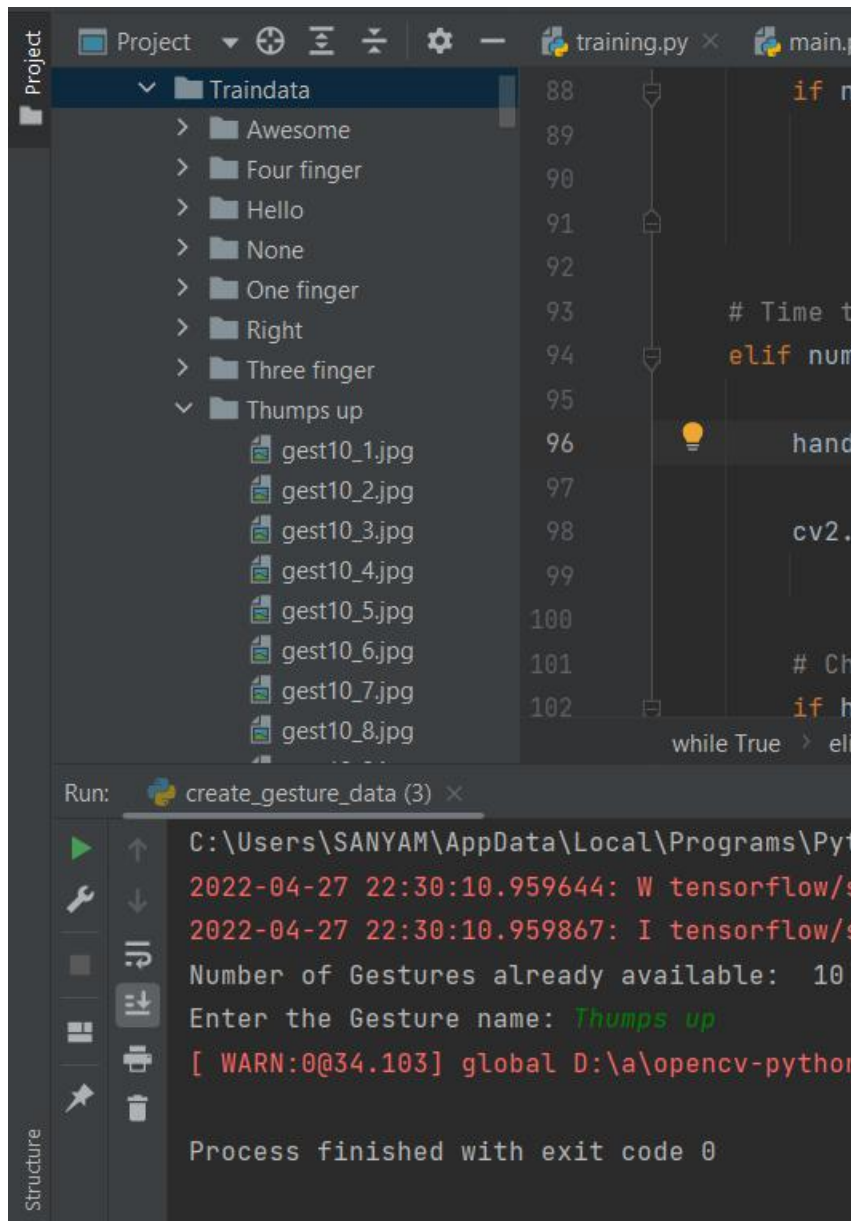


```
Run: create_gesture_data (3)
C:\Users\SARYAM\AppData\Local\Programs\Python\Python39\python.exe E:\PyCharm\PycharmProjects\SC-Project\Different-Hand-Gesture-Recognition-Using-CNN\create_gesture_data.py
2022-04-27 22:30:10.959644: W tensorflow/stream_executor/platform/default/dso_loader.cc:64] Could not load dynamic library 'cudart64_110.dll'; dlerror: cudart64_110.dll not found
2022-04-27 22:30:10.959867: I tensorflow/stream_executor/cuda/cudart_stub.cc:29] Ignore above cudart dlerror if you do not have a GPU set up on your machine.
Number of Gestures already available: 10
Enter the Gesture name: Thumbs up
[ WARN:0054,105] global 0:\opencv-python\opencv-python\opencv\modules\videoio\src\cap_msmf.cpp (539) 'anonymous-namespace':::SourceReaderCB::~SourceReaderCB terminating async ca
Process finished with exit code 0
```

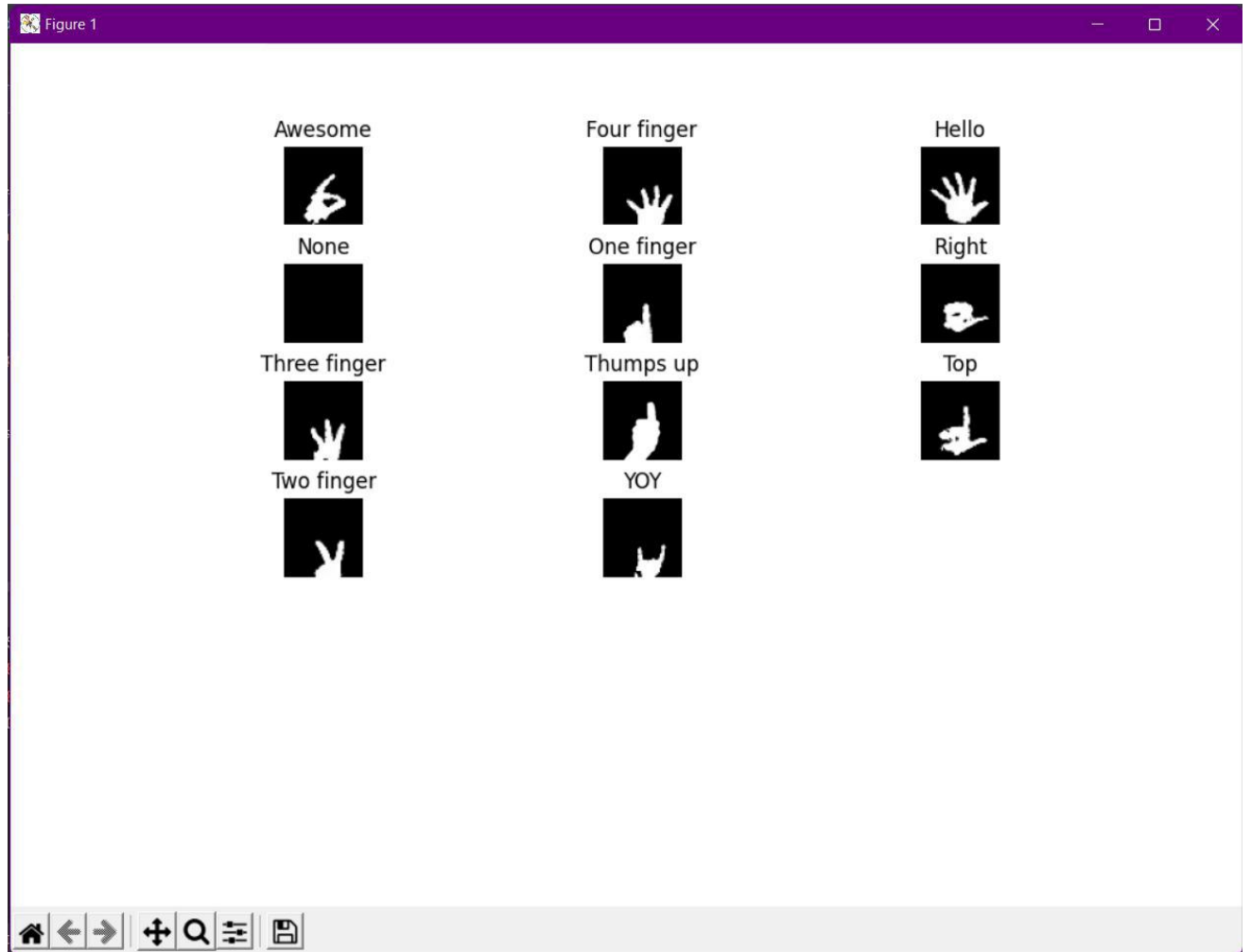


The images will be simultaneous added in the folder newly created by the program.





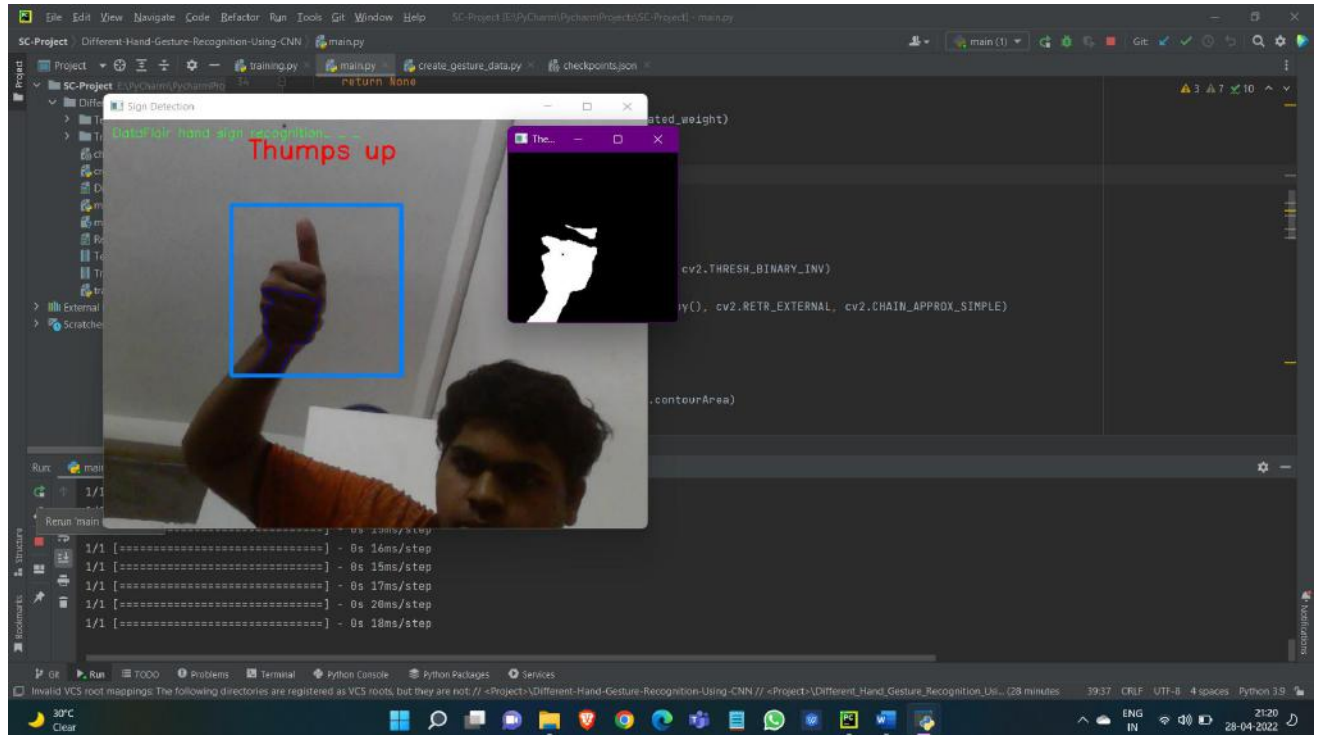
Now let's train the model and try out with this new gesture.



```

training
Epoch 1/10
1030/1030 [=====] - 79s 67ms/step - loss: 0.4386 - accuracy: 0.8842 - val_loss: 0.6872 - val_accuracy: 0.8173 - lr: 0.0010
Epoch 2/10
1030/1030 [=====] - 28s 27ms/step - loss: 0.0431 - accuracy: 0.9875 - val_loss: 0.4679 - val_accuracy: 0.8936 - lr: 0.0010
Epoch 3/10
1030/1030 [=====] - 38s 29ms/step - loss: 0.0154 - accuracy: 0.9966 - val_loss: 0.5271 - val_accuracy: 0.8916 - lr: 0.0010
Epoch 4/10
1030/1030 [=====] - 28s 27ms/step - loss: 0.0046 - accuracy: 0.9991 - val_loss: 0.5164 - val_accuracy: 0.8855 - lr: 5.0000e-04
['loss', 'accuracy'] [0.020932118026478767, 1.0]
loss of 0.020932118026478767; accuracy of 100.0%
{'loss': [0.4386160671710968, 0.04311235994100571, 0.015394957736134529, 0.004579574801027775], 'accuracy': [0.8841747641563416, 0.9874757528305054, 0.996601939201355, 0.99912619]
loss of 0.5448143482208252; accuracy of 69.9999988879071%

```



## References:

- [1] Hand Gesture Recognition Using CNN for Post-Stroke People, 2019, Norah Alnaim, Abdulrahman Albar, and Maysam F Abbod
- [2] Static Hand Gesture Recognition using Convolutional Neural Network with Data Augmentation, 2019, Md. Zahirul Islam, Mohammad Shahadat Hossain, Raihan UI Islam, and Karl Andersson
- [3] IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition, 2020, Gibran Benitez-Garcia, Jesus Olivares-Mercado , Gabriel Sanchez-Perez, and Keiji Yanai
- [4] Real-time Hand Gesture Detection and Classification Using Convolutional Neural Networks, 2019, Okan Kopuklu, Ahmet Gunduz , Neslihan Kose , and Gerhard Rigoll
- [5] Hand Recognition Using Geometric Classifiers, 2004, Yaroslav Bulatov, Sachin Jambawalikar, Piyush Kumar, and Saurabh Sethia
- [6] IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition, 2020, Gibran Benitez-Garcia, Jesus Olivares-Mercado , Gabriel Sanchez-Perez, and Keiji Yanai
- [7] Real-Time Hand Tracking and Gesture Recognition System-2009  
Nguyen Dang Binh, Enokida Shuichi
- [8] Development of hand gesture recognition system using machine learning(2020)-  
Priyanka Parvathy Kamalraj Subramaniam

[9] "Real-time hand tracking and gesture recognition system using neural networks."-2009  
Maung, Tin Hninn Hninn

[10] "An automatic hand gesture recognition system based on Viola-Jones method  
and SVMs."- Yun, Liu, and Zhang Peng.



# IPN Hand: A Video Dataset and Benchmark for Real-Time Continuous Hand Gesture Recognition

Gibran Benitez-Garcia <sup>\*</sup>, Jesus Olivares-Mercado<sup>†</sup>, Gabriel Sanchez-Perez <sup>†</sup> and Keiji Yanai<sup>\*</sup>

<sup>\*</sup>Department of Informatics, The University of Electro-Communications, Tokyo, Japan

Email: gibran@ieee.org, yanai@cs.uec.ac.jp

<sup>†</sup>Instituto Politecnico Nacional, ESIME Culhuacan, Mexico City, Mexico

Email: jolivares@ipn.mx, gasanchezp@ipn.mx

**Abstract**—Continuous hand gesture recognition (HGR) is an essential part of human-computer interaction with a wide range of applications in the automotive sector, consumer electronics, home automation, and others. In recent years, accurate and efficient deep learning models have been proposed for HGR. However, in the research community, the current publicly available datasets lack real-world elements needed to build responsive and efficient HGR systems. In this paper, we introduce a new benchmark dataset named IPN Hand with sufficient size, variety, and real-world elements able to train and evaluate deep neural networks. This dataset contains more than 4,000 gesture samples and 800,000 RGB frames from 50 distinct subjects. We design 13 different static and dynamic gestures focused on interaction with touchless screens. We especially consider the scenario when continuous gestures are performed without transition states, and when subjects perform natural movements with their hands as non-gesture actions. Gestures were collected from about 30 diverse scenes, with real-world variation in background and illumination. With our dataset, the performance of three 3D-CNN models is evaluated on the tasks of isolated and continuous real-time HGR. Furthermore, we analyze the possibility of increasing the recognition accuracy by adding multiple modalities derived from RGB frames, i.e., optical flow and semantic segmentation, while keeping the real-time performance of the 3D-CNN model. Our empirical study also provides a comparison with the publicly available nvGesture (NVIDIA) dataset. The experimental results show that the state-of-the-art ResNext-101 model decreases about 30% accuracy when using our real-world dataset, demonstrating that the IPN Hand dataset can be used as a benchmark, and may help the community to step forward in the continuous HGR. Our dataset and pre-trained models used in the evaluation are publicly available at [github.com/GibranBenitez/IPN-hand](https://github.com/GibranBenitez/IPN-hand).

## I. INTRODUCTION

Gestures are a natural form of human communication [1]. Hand gesture recognition (HGR) is an essential part of human-computer interaction. Systems using vision-based interaction and control are more common nowadays [2]–[4]. Compared to traditional inputs such as mouse and keyboard, vision-based interfaces can be more practical and natural based on the intuitiveness of designed gestures. HGR has a wide range of applications in the automotive sector, consumer electronics, public transit, gaming, home automation, and others [4]–[6]. For these applications, HGR systems must be designed to work online and deal with continuous gestures that users may input. There are mainly four issues that systems must deal with for continuous HGR applications: (i) continuous gestures without transition states, (ii) natural behaviors of

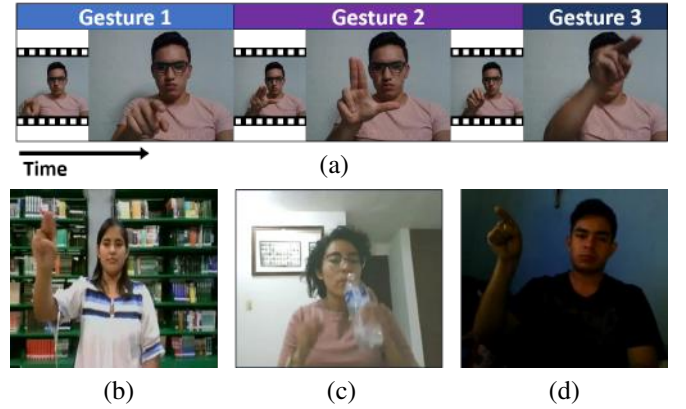


Fig. 1. Some examples that show the challenges of our dataset. (a) Continuous gestures without transition states. (b) Clutter backgrounds. (c) Natural interaction with objects. (d) Weak illumination conditions.

users' hands similar to target gestures, (iii) intra-class variability of gestures' duration, and (iv) lag between performing a gesture and its classification. It is worth mentioning that (i) is particularly important for some real-world applications such as interacting with touchless screens. For example, someone can open a picture and zoom in, by using two continuous gestures (double-click + zoom-in) without a transition state that forces the hand returning to a neutral position before the second gesture. This example is shown in Figure 1(a).

Thanks to the advances on deep neural networks, in recent years, accurate and efficient deep learning models have been proposed to overcome the challenges on continuous HGR [7]–[11]. However, these methods were evaluated using datasets that do not cover the main real-world issues. Currently, it is not easy to find a benchmark dataset able to evaluate the four main issues of continuous HGR. Most of the hand gesture dataset used for continuous HGR, like ChaLearn ConGD [12], nvGesture [7] and EgoGesture [13], do not include continuous gestures without transition states (i), nor natural hand movements as non-gesture actions (ii). To the best of our knowledge, there is no publicly available hand gesture datasets that cover these two issues of continuous HGR. Note that, some works have designed specific datasets for controlling automotive interfaces [14], [15], that partially include these issues. However, the datasets are not publicly available.

TABLE I  
COMPARISON OF THE PUBLIC CONTINUOUS GESTURE DATASETS

Dataset	Instances	Videos	Instance/video	Classes	Subjects	Scenes	View	Modalities
ChaLearn ConGD, 2016 [12]	47,933	22,535	2.1	249	21	15	3rd	RGB-D
nvGesture, 2016 [7]	1,532	1,532	1.0	25	20	1	3rd	RGB-D
EgoGesture, 2018 [13]	24,161	2,081	11.6	83	50	6	1st	RGB-D
IPN Hand (ours)	4,218	200	21.1	13	50	28	3rd	RGB

In this paper, we introduce a new dataset called IPN Hand for the task of continuous hand gesture recognition. The dataset contains more than four thousand RGB gesture samples and 800 thousand frames from 50 distinct subjects. We design 13 classes of static and dynamic gestures for interaction with touchless screens. Our dataset has the most realistic scenario for continuous HGR than other hand gesture datasets. IPN Hand includes the largest number of continuous gestures per video and the largest speed of intra-class variation for different subjects when they were performing the same gesture. Besides, our dataset is more complex as it was collected from about thirty representative scenes with considerable variation, including clutter backgrounds, strong and weak illumination conditions, static and dynamic background environments. We specially design two scenarios that reflects the real-world issues of continuous HGR: when continuous gestures are performed without transition states, and when subjects perform natural movements with their hands as non-gesture actions. Some examples of the main challenges of our dataset are shown in Figure 1.

Given our dataset, we evaluate three state-of-the-art (SOTA) methods based on 3D-CNN models for the tasks of isolated and continuous real-time HGR. Furthermore, we analyze the possibility of increasing the recognition accuracy by adding multiple modalities derived from RGB frames. Specifically, we evaluate the data level fusion of RGB with semantic segmentation of target hands as an alternative of the RGB+Optical Flow or RGB+Depth modalities for real-time HGR. Our empirical study also provides a comparison with the publicly available nvGesture (NVIDIA) dataset. The experimental results on the IPN Hand dataset demonstrate that it can be used as a benchmark for the data-hungry 3D-CNN methods, which may help the community to step forward in the continuous HGR.

## II. RELATED WORKS

### A. Datasets for continuous HGR

Existing continuous HGR datasets differ by factors such as scale, the number of classes, sensors used, and the domain of gestures. However, all of them must include non-gesture frames to emulate the online behavior of real applications. These frames are important to define the realism of the designed dataset. Therefore, we analyze the commonly used datasets based on this element.

The ChaLearn LAP ConGD dataset [12] is derived from 9 different gesture domains, from Italian sign language, activities to pantomime. It contains 249 classes and more than 40 thousand instances from 21 subjects, which makes ChaLearn ConGD the largest dataset for continuous HGR. This dataset



Fig. 2. Comparison of gesture vs. non-gesture frames from the public continuous gesture datasets. Note that only the first row (blue) shows examples of gesture frames.

contains videos with one to five continuous gestures with transition states. The nvGesture dataset [7] is designed to control in-car automotive devices, and it includes 25 gesture types from 20 subjects, consisting of 1 532 instances. Note that this dataset is commonly used for online HGR evaluation, even though it only contains videos with isolated gestures. The EgoGesture dataset [13] is a benchmark dataset for egocentric (first-person) view, which consists of 83 classes with more than 20 thousand instances from 50 subjects. This dataset includes at max 12 instances per video. To the best of our knowledge, our proposed IPN Hand dataset contains the largest number of instances per video (21), and different scenes (28), making it suitable for evaluating continuous HGR systems. Detailed comparison between our IPN Hand and related gesture datasets can be found in Table I.

Figure 2 shows the comparison of the non-gesture frames from the public continuous gesture datasets. Most of the non-gesture frames from the ChaLearn ConGD dataset consist of transition states between gestures, usually showing the subject in a neutral position as in the first column of the figure. On the other hand, the nvGesture shows drivers with their hands at the steering wheel, while EgoGesture shows only backgrounds with the hands out of view as non-gesture frames. It is clear that our dataset presents more challenge on distinguishing gesture vs. non-gesture frames, since we include subjects performing natural movements with their hands.

### B. Continuous HGR

Continuous HGR can be divided into two stages: gesture spotting and gesture classification. Gesture spotting aims to detect temporal video segments which contain gesture instances,

while the classification stage aims to classify the gesture of each spotted segment. Multi-stream architectures have been widely employed for both tasks. Simonyan et al. [16] were the pioneers of fusing features from two modalities, using one stream with RGB images and the other with flow fields for isolated HGR. On This multi-modality approach is prevalent as well for continuous HGR, as shown in the 2017 ChaLearn Look At People (LAP) continuous gesture detection (ConGD) challenge [17], where all entries used multi-stream architectures of at least RGB and depth. For instance, the winners [8] introduced a two-stream 3D-CNN combining hand-location features of RGB and depth modalities by explicitly overlaying a black mask on the input frames. They firstly spotted the gestures based on a dataset-specific observation: subjects raise their hands at the beginning of gestures and put them down again at the end. One year after ConGD challenge, Zhu et al. [9] overcame the results of the winners by proposing a temporal dilated 3D-CNN architecture to binary classify gesture/non-gesture frames, and 3D-CNN+LSTM for gesture classification. The current SOTA method of ConGD uses a 12-channels architecture with extra RNN layers to simultaneously spot and classify continuous gestures [10]. Recently, Kopuklu et al. [11] proposed a hierarchical structure of 3D-CNN architectures to detect and classify continuous hand gestures. Their spotting method consists of detecting gesture frames using a shallow 3D-CNN model on eight consecutive frames, while a SOTA model is used for classification only if a gesture is detected.

To evaluate our dataset for real-world applications of continuous HGR, detection and classification must work online or even with a negative lag between performing a gesture and its classification feedback. Therefore, we follow the hierarchical structure of 3D-CNNs [11], since it can detect a gesture when a confidence measure reaches a threshold level before the gesture ends (early-detection). Furthermore, we evaluate the multi-modality accuracy derived from RGB frames, by keeping in mind the real-time performance. We employ a data level fusion to avoid the significant increase in the computational cost of the 3D-CNN models. Besides, we propose to use semantic segmentation results as an alternative to the absent depth modality and the computational expensive optical flow.

### III. THE IPN HAND DATASET

#### A. Data collection

To collect the dataset, the RGB videos were recorded in the resolution of  $640 \times 480$  with the frame rate of 30 fps. The participants were asked to record the gestures using their own PC or laptop by keeping the defined resolution and frame rate. Thus, the distance between the camera and each subject varies, since we instructed all participants to be located in a comfortable position to manipulate the screen of their PC with hand gestures. In this way, the videos were generated with 28 different scenes in total. For some participants that do not have access to a camera that can cover the recording specifications, we prepared three scenes (scene 2-4, shown in Figure 3(b)-(d)), including clutter and variable illumination



Fig. 3. Some examples of the scenes with more videos in the dataset. Note that all subjects are developing the same gesture, "pointing with two fingers".

backgrounds. When collecting data, we first teach the subjects how to perform each gesture and tell them the gesture names (short descriptions). Then we generate four gesture name lists with random order for recording four videos per subject. Thus, the subject was told the gesture name and performed the gesture accordingly. They were asked to continuously perform 21 gestures with 3 random breaks as a single session which was recorded as a video.

#### B. Dataset characteristics

1) *Gesture Classes*: We design the gestures in our dataset focused on interaction with touchless screens. We include gestures able to control the location of the pointer on the screen (pointer), and to manipulate interfaces (actions). We take some gestures used in common smartphones, as they are natural and easy to remember by the users. Thus, we designed 13 gestures (shown in Figure 4) defined to control the punter and actions.

For the punter location, we defined two static gestures of pointing to the screen with one and two fingers, respectively. Note that these gestures are static due to the gesture itself does not need temporal information to be detected. However, the application of this gesture includes the hand movement to control the pointer. For the action gestures, we defined 11 gestures, including click with one and two fingers, throw to four positions (up, down, left & right), double click with one and two fingers, zoom-in, and zoom-out. Table II provides the name and descriptions of each gesture in our dataset. Note that we also include segments were natural hand movements are performed as non-gestures states.

2) *Subjects*: A small number of subjects could make the intra-class variation very limited. Hence, we asked 50 subjects for our data collection. In the 50 subjects, there are 16 females and 34 males. The average age of the subjects is 21.2, where the minimum age is 18, and the maximum age is 25. All of the subjects are currently students at Instituto Politecnico from Mexico.

3) *Extreme illumination and clutter backgrounds*: To evaluate the robustness to the illumination change of the baseline



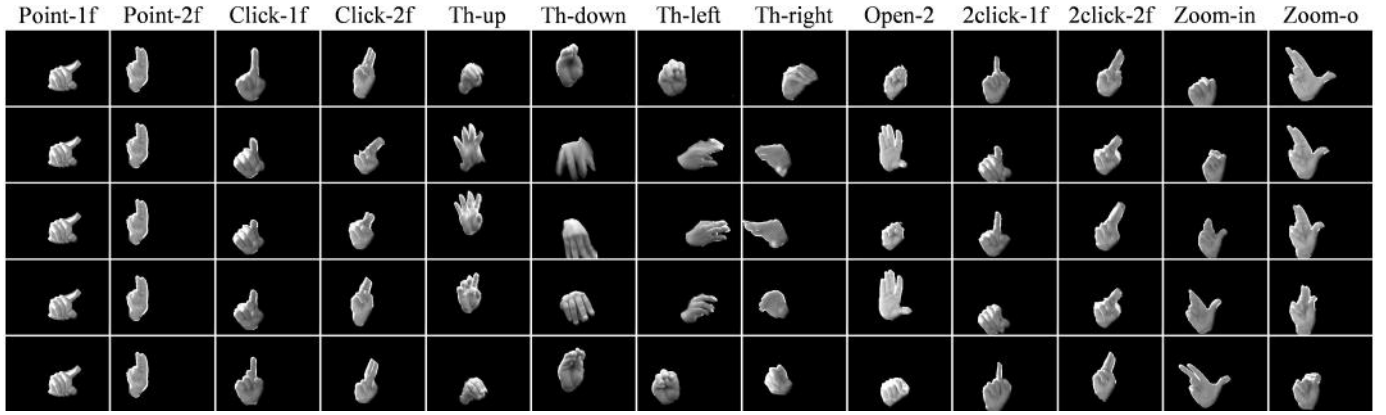


Fig. 4. Examples of the 13 gesture classes included in our dataset. For visualization purposes, semantic segmentation masks were blended to the RGB images.

TABLE II  
STATISTICS PER GESTURE OF OUR IPN HAND DATASET

id	Gesture	Name	Instances	Mean duration ( <i>std</i> )	Duration $\overline{std}_k$
0	No gesture	No-gest	1431	147 (133)	0.904
1	Pointing with one finger	Point-1f	1010	219 (67)	0.308
2	Pointing with two fingers	Point-2f	1007	224 (69)	0.309
3	Click with one finger	Click-1f	200	56 (29)	0.517
4	Click with two fingers	Click-2f	200	60 (43)	0.718
5	Throw up	Th-up	200	62 (25)	0.400
6	Throw down	Th-down	201	65 (28)	0.424
7	Throw left	Th-left	200	66 (27)	0.400
8	Throw right	Th-right	200	64 (28)	0.439
9	Open twice	Open-2	200	76 (31)	0.410
10	Double click with one finger	2click-1f	200	68 (28)	0.412
11	Double click with two fingers	2click-2f	200	70 (30)	0.435
12	Zoom in	Zoom-in	200	65 (29)	0.440
13	Zoom out	Zoom-o	200	64 (28)	0.432

methods, we have data collected under extreme conditions such as facing a window with strong sunlight (Figure 3(b)), or in a room with almost null artificial light (Figure 1(d)). Some scenes with static background placed with student-life stuff (Figure 1(b)), and dynamic background with walking people appearing in the camera view (Figure 3(d)-(f)) were also included.

### C. Dataset statics

Fifty distinct subjects participated in performing 13 classes of gestures in 28 diverse scenes. Totally 4,218 gesture instances and 800,491 frames were collected in RGB. Figure 5 illustrates the sample distribution on each video among the 28

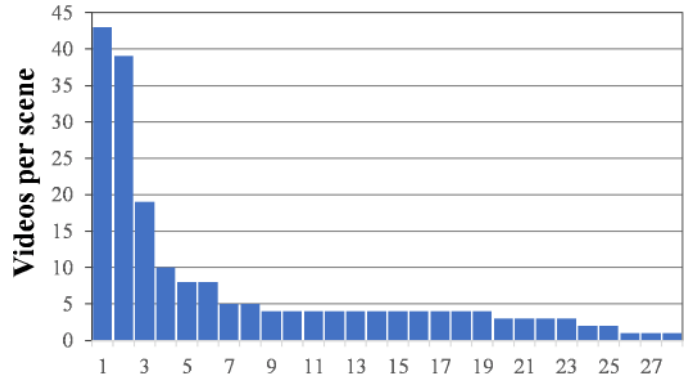


Fig. 5. Distribution of the 28 different scenes included in the dataset.

scenes. In the figure, the horizontal axis and the vertical axis indicate the ID of the scenes and the number of the videos, respectively. In addition, Figure 3 shows examples of the six scenes with more videos on our dataset.

When data collection, 21 gestures are considered as a session and recorded in a single video. Thus, we get 200 RGB videos in total. Note that the order of the gestures performed is semi-randomly generated, trying to include one dynamic after one static gesture, which resembles the realistic interaction with a touchless screen. The start and end frame index of each gesture instance in the video were manually labeled, which provides enough information to train and evaluate continuous HGR approaches in a fully supervised manner. In the dataset, the minimum length of a gesture is 9 frames. The maximum length of a gesture is 650 frames.

In Table III, we show the data statistics of our dataset compared with the continuous HGR datasets that are currently publicly available. The dataset statistics include the number of total frames, the mean video durations, the mean of the gesture sample durations, the standard deviation of gesture durations, the percentage of the training data. The mean and standard deviation of the gesture sample durations is calculated over the samples from all gesture classes in the dataset. Following [13], we use the normalized standard deviation

TABLE III  
STATISTICS OF THE PUBLIC CONTINUOUS GESTURE DATASETS

Dataset	Frames	Mean video duration	Mean gesture duration	Duration std	Duration $nstd_k$	% of train
ChaLearn ConGD, 2016 [12]	1,714,629	76.1	41	18.5	0.37	0.635
nvGesture, 2016 [7]	122,560	80.0	71	32.3	0.3	0.685
EgoGesture, 2018 [13]	2,953,224	1,419.1	38	13.9	0.33	0.595
IPN Hand (ours)	800,491	4,002.5	140	93.9	0.43	0.739

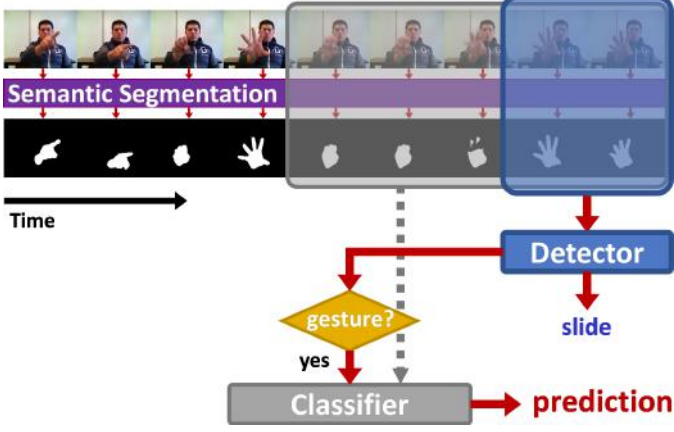


Fig. 6. The flowchart of the continuous HGR approach based on two hierarchical 3D-CNNs models.

for gesture duration in each gesture class to describe the speed variation of different subjects when performing the same gesture. The normalized standard deviation of durations in a gesture class  $k$  is calculated as follows:

$$nstd_k = \frac{1}{l^k} \sqrt{\frac{\sum_i^N (l_i^k - l^k)^2}{N}} \quad (1)$$

where in gesture class  $k$ ,  $l_i^k$  represents the duration of the  $i$ th sample,  $l^k$  is the average duration of samples, and  $N$  is the number of samples. We get the average  $nstd_k$  over all gesture classes, for the whole dataset.

From Table III, we can see that our IPN Hand dataset has the largest duration  $nstd_k$  (0.43), significantly larger than EgoGesture, which is the second largest. This demonstrates that our dataset has a large speed variation for different subjects when performing the same gesture. In resume, our proposed dataset covers all the real-world issues for continuous HGR, as described in the introduction: (i) continuous gestures without transition states, (ii) natural behaviors of users' hands similar to target gestures, (iii) intra-class variability of gestures' duration ( $nstd_k$ ).

#### IV. BENCHMARK EVALUATION

We evaluate three SOTA 3D-CNN models as baselines for the tasks of isolated and continuous HGR, with our new IPN Hand dataset. For continuous HGR, we adopt a two-model hierarchical architecture to detect and classify the continuous input stream of frames.

##### A. Hierarchical 3D-CNNs for continuous HGR

As mentioned before, we use the framework proposed by Kopuklu et al. [11] to detect and classify continuous hand gestures. The flowchart of the two-model hierarchical architecture is shown in Figure 6. Sliding windows with a constant stride run through incoming video frames where the detector queue is placed at the very beginning of the classifier queue. If the detector recognizes a gesture, then the classifier is activated. The detector's output is post-processed for more robust performance, and the final decision is made using a single-time activation block where only one activation occurs per performed gesture. Note that the sliding window of the detector is the only who is working every single stride. In contrast, the classification window is activated based on detectors output, so the inference time is not always the sum of both models.

Since we also evaluate multi-modal 3D-CNNs models, we first calculate the semantic segmentation masks of each streaming frame. For this task, we use the efficient semantic segmentation approach of HarDNet (Harmonic Dense Net) [18] which achieves SOTA results with a network inspired by DenseNet [19]. Its core component, the *HarDBlock* (Harmonic Dense Block), is specifically designed to address the problem of the GPU memory traffic. Therefore, we can achieve real-time performance on each frame before applying 3D-CNNs models.

##### B. Experimental setup

We randomly split the data by subject into training (74%), and testing (26%) sets, resulting in 148 training, and 52 testing videos. The numbers of gesture instances in training, and testing splits are 3 117, and 1 101, respectively. Thus, 37 and 13 subjects were designed for training and testing, respectively.

1) *Implementation details*: We compare the 3D-CNN models of C3D [20], and the 3D versions [21] of Resnet-50 [22] and ResNeXt-101 [23], as deep models for classification. On the other hand, for the shallow 3D-CNN detector we use the ResLight (Resnet-10) as proposed in [11]. All 3D-CNN models were trained using a fully supervised strategy using the manually segmented gestures.

For the multi-modality tests, we train a HarDNet [18] model, and use the SPyNet [24] approach for calculation of semantic segmentation and optical flow, respectively. We trained the HarDNet model with a synthetic hand pose estimation dataset [25] which contains more than 40 thousand images with hands fully annotated at pixel-level<sup>1</sup>. For the SPyNet, we

<sup>1</sup><https://lmb.informatik.uni-freiburg.de/projects/hand3d/>

TABLE IV  
RESULTS OF ISOLATED HGR TASK USING OUR IPN HAND DATASET

Model	Input sequence	Modality	Results	Parameters	Model Size	Inference time
C3D	32-frames	RGB	77.75	50.75 M	387 MB	76.2 ms
ResNeXt-101	32-frames	RGB	83.59	47.51 M	363 MB	27.7 ms
ResNeXt-101	32-frames	RGB-Flow	<b>86.32</b>	47.56 M	363 MB	28.9 ms
ResNeXt-101	32-frames	RGB-Seg	84.77	47.56 M	363 MB	28.9 ms
ResNet-50	32-frames	RGB	73.1	<b>46.25 M</b>	<b>353 MB</b>	<b>17.8 ms</b>
ResNet-50	32-frames	RGB-Flow	74.65	46.27 M	<b>353 MB</b>	18.2 ms
ResNet-50	32-frames	RGB-Seg	75.11	46.27 M	<b>353 MB</b>	18.2 ms

TABLE V  
COMPARISON OF THE EXTRA PROCESSES FOR MULTI-MODALITY MODELS.

Process	Model	Params.	Model size	Inference time
Segmentation	HardNet	4.114 M	15.8 MB	<b>8.1 ms</b>
Optical Flow	SPyNet	<b>1.440 M</b>	<b>5.50 MB</b>	21.9 ms

used the open-source implementation and pre-trained model of [26] to calculate real-time densely optical flow from each input frame.

All 3D-CNN models were pre-trained on the Jester dataset [27], while the HardNet on the ImageNet dataset [28]. Contrary, since ResLight is a compact model, we trained it from scratch using all non-gesture and gesture instances of the IPN Hand training set. The inference time (FPS) was measured on an Intel Core i7-9700K desktop with a single NVIDIA GTX 1080ti GPU. More implementation details, as well as training and evaluation codes can be found in our open-source repository<sup>2</sup>

### C. Isolated HGR task

We evaluate this task with the conventional classification metric. We segment the video sequences into isolated gesture samples based on the beginning and ending frames manually annotated. The learning task is to predict class labels for each gesture sample. We use classification accuracy, which is the percent of correctly labeled examples, and the confusion matrix of the predictions, as evaluation metrics for this learning task.

#### 1) Experimental results using the IPN Hand dataset:

Table IV presents the results of evaluated models with different modalities using our IPN Hand dataset. As expected, the best results were obtained by the ResNeXt-101 model, which can barely achieve real-time performance with the different modalities. However, it is clearly faster and more accurate than the robust C3D model. On the other hand, ResNet-50 is the most efficient model, but present the lowest accuracy results among the evaluated approaches.

It is worth noting that our RGB-seg achieves competitive results compared to the RGB-flow, which is significant since the optical flow process is more computationally expensive. To evaluate the efficiency of the multi-modal approaches, Table V shows the inference time and the model size of extra-processes related to semantic segmentation and optical flow estimation.

<sup>2</sup><https://github.com/GibranBenitez/IPN-hand>

	P1	P2	C1	C2	T-u	T-d	T-l	T-r	O2	2c1	2c2	Z-i	Z-o
Point-1f	92	6	0	0	1	0	0	1	0	0	0	0	0
Point-2f	4	95	0	0	0	0	0	1	0	0	0	0	0
Click-1f	4	4	73	0	0	2	0	0	0	13	0	2	2
Click-2f	0	6	0	63	0	0	0	4	0	2	21	0	4
Th-up	2	2	0	0	85	2	0	0	10	0	0	0	0
Th-down	4	0	0	0	2	92	0	0	2	0	0	0	0
Th-left	0	0	0	0	0	0	94	2	2	0	0	0	2
Th-right	2	0	0	0	0	2	0	96	0	0	0	0	0
Open-2	0	0	0	0	2	0	0	0	87	0	0	12	0
2click-1f	6	0	31	0	0	0	0	0	0	60	2	2	0
2click-2f	0	4	0	33	0	0	0	0	0	2	60	2	0
Zoom-in	0	0	0	0	0	0	2	2	0	0	0	90	6
Zoom-out	0	4	2	4	0	0	0	4	0	0	0	8	79

Fig. 7. Confusion matrix of the best result obtained from ResNext-101 model with RGB-flow.

TABLE VI  
RESULTS OF ISOLATED HGR TASK USING THE NVGESTURE DATASET

Model	Input sequence	Modality	Results
ResNeXt-101	32-frames	RGB	79.46
<b>ResNeXt-101</b>	<b>32-frames</b>	<b>RGB-Flow</b>	<b>82.36</b>
ResNeXt-101	32-frames	RGB-Seg	82.15

From this table, we can see that the semantic segmentation is more than two times faster than the optical flow, making the RGB-seg alternative feasible for real-time applications.

In addition, in the Figure 7, we show the confusion matrix of the best result obtained for isolated HGR, ResNext-101 with RGB-flow. As expected, the problems are related to the gestures that are closer such as clicks with double clicks.

2) *Experimental results using the nvGesture dataset:* As mentioned before, we also evaluate the performance of our RGB-seg alternative with a common online HGR dataset. Table VI presents the results of the evaluated models with different modalities using the nvGesture dataset. From these results, we can notice that the trend of the multi-modal results from IPN Hand is still valid (RGB-flow > RGB-seg > RGB).

In addition, Table VII shows the result of our real-time RGB-seg alternative compared to SOTA methods of nvGesture dataset. From this table, we validate the use of semantic segmentation as an important source of valuable features for HGR. The results of ResNeXt-101 with RGB-seg, are competitive to SOTA methods that even employ different modalities, such as depth. These findings are significant due to the semantic segmentation process only takes 8 ms with an

TABLE VII  
COMPARISON WITH SOTA METHODS OF NVGESTURE DATASET.

Model	Modality	Results
C3D [20]	RGB	73.8
R3DCNN [7]	RGB	74.1
MTUT [29]	RGB*	81.3
R3DCNN [7]	RGB+Flow	79.3
<b>MFF [30]</b>	<b>RGB+Flow</b>	<b>84.7</b>
R3DCNN [7]	Depth+Flow	82.4
ResNeXt-101	RGB+Seg	82.2

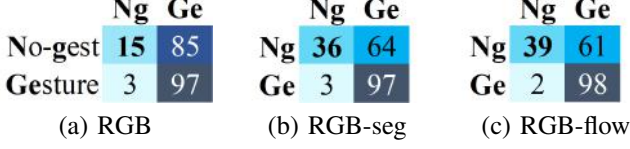


Fig. 8. Confusion matrices of gesture detection on the IPN hand dataset using different modalities for the Resnet-Light model.

input image of  $360 \times 240$  pixels.

#### D. Continuous HGR task

We use the Levenshtein accuracy [11] as evaluation metric for the continuous HGR tasks. This metric employs the Levenshtein distance to measure the distance between sequences by counting the number of item-level changes (insertion, deletion, or substitutions) to transform one sequence into the other. In the case of continuous hand gestures, the difference between the sequence of predicted and ground truth gestures is measured. For example, if a ground truth sequence is  $[1, 2, 3, 4, 5, 6, 7, 8, 9]$  and predicted gestures of a video is  $[1, 2, 7, 4, 5, 6, 6, 7, 8, 9]$ , the Levenshtein distance is 2. Defined by the deletion of one of the "6" which is detected two times, and the substitution of "7" with "3". Thus, the Levenshtein accuracy is obtained by averaging this distance over the number of true target classes. In our example, the accuracy is  $1 - (2/9) \times 100 = 77.78\%$ .

We obtain the average Levenshtein accuracy over the 52 testing videos to assess the continuous HGR performance. Besides, we also evaluate the detection accuracy of the 3D-CNN detectors with different multi-modalities. Note that detectors are trained and evaluated using isolated gesture vs. non-gesture samples. So that we use a binary classification accuracy, and the confusion matrix of the predictions, as evaluation metrics for these models.

1) *Experimental results:* First, we evaluate the detector model with the different modalities in Table VIII. As expected, the same trend is maintained, but since ResLight-10 is a much more compact model, the inference time is not affected. It is also important to analyze the misclassification of the detector, therefore Figure 8 shows the confusion matrices from each modality. It is clear that the benefits from multi-modal approaches are reflected in the detection of non-gesture frames, since the RGB approach misrecognized 85% of these frames.

We evaluate the complete process of the hierarchical two-model approach using Resnet-50 and ResNeXt-101 as classifiers, and ResLight-10 as a detector. In addition, we also

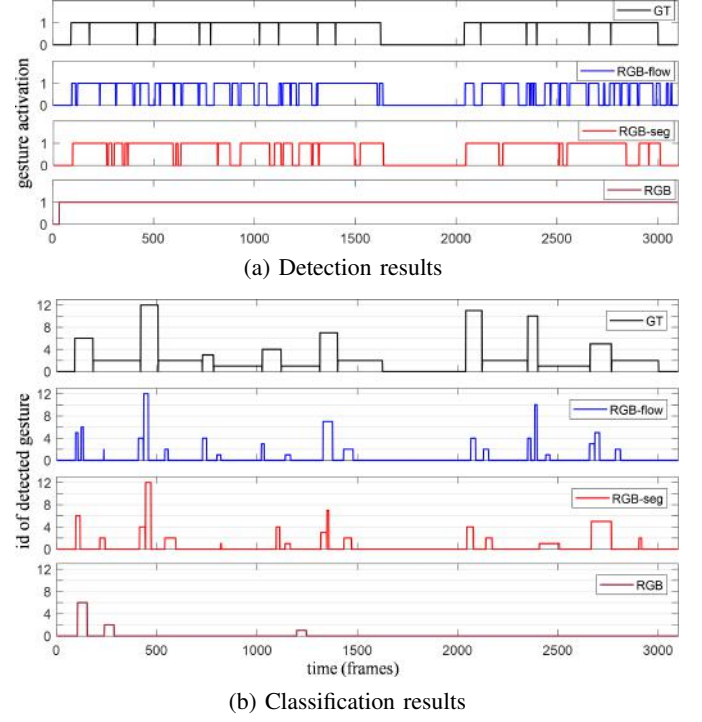


Fig. 9. Qualitative results of the proposed networks. The most significant improvements occur on pixels belonging to large objects.

analyze the multi-modal alternatives covering real-time performance. Table IX shows the Levenshtein accuracy, as well as the model size and inference time of each evaluated model and modality. The results of this test make more evident the advantages of using the RGB-seg alternative for real-time continuous HGR. For instance, the inference time results of ResNeXt-101 RGB are comparable with those of ResNet-50 RGB-seg. However, the Levenshtein accuracy of the latter is significantly better (around 10% of improvement). Finally, we show the temporal visualization of continuous predictions from one testing video in Figure 9.

#### V. CONCLUSION

In this paper, we introduced a new benchmark dataset for continuous HGR that includes real-world issues, such as continuous gestures without transition states, natural behaviors of users' hands, and large intra-class variability of gestures' duration. Besides, we evaluate the data level fusion of RGB with semantic segmentation as an alternative of the RGB+Optical Flow or RGB+Depth modalities for real-time HGR. From the experimental results, we conclude that RGB-seg is a suitable multi-modal alternative for real-time continuous hand gesture recognition. Furthermore, we believe that the proposed dataset can be used as a benchmark and help the community to move steps forward on the continuous HGR.

#### ACKNOWLEDGMENT

This work was supported by JSPS KAKENHI Grant Number 15H05915, 17H01745, 17H06100 and 19H04929.



TABLE VIII  
RESULTS OF THE DETECTOR MODEL USING OUR IPN HAND DATASET

Model	Input sequence	Modality	Results	Parameters	Model Size	Inference time
ResLight-10	8-frames	RGB	75.4	<b>0.895 M</b>	<b>6.83 MB</b>	<b>2.4 ms</b>
ResLight-10	8-frames	RGB-Flow	<b>82.43</b>	0.908 M	6.94 MB	2.9 ms
ResLight-10	8-frames	RGB-Seg	80.06	0.908 M	6.94 MB	2.9 ms

TABLE IX  
RESULTS OF THE HIERARCHICAL TWO-MODEL APPROACH FOR CONTINUOUS HGR USING OUR IPN HAND DATASET

Model	Modality	Results	Model size			Inference time		
			detector	classifier	total	detector	classifier	total
ResNeXt-101	RGB	25.34	<b>6.83 MB</b>	<b>363 MB</b>	<b>370 MB</b>	<b>2.9 ms</b>	<b>27.7 ms</b>	<b>30.1 ms</b>
ResNeXt-101	RGB-Flow	<b>42.47</b>	12.4 MB	363 MB	375 MB	11.1 ms	28.9 ms	53.7 ms
ResNeXt-101	RGB-seg	39.01	22.7 MB	363 MB	386 MB	24.8 ms	28.9 ms	39.9 ms
Resnet-50	RGB	19.78	<b>6.83 MB</b>	<b>353 MB</b>	<b>360 MB</b>	<b>2.9 ms</b>	<b>17.8 ms</b>	<b>20.4 ms</b>
Resnet-50	RGB-Flow	<b>39.47</b>	12.4 MB	353 MB	365 MB	11.1 ms	18.2 ms	43.1 ms
Resnet-50	RGB-seg	33.27	22.7 MB	353 MB	376 MB	24.8 ms	18.2 ms	29.2 ms

## REFERENCES

- [1] A. Kendon, "Gesticulation and speech: Two aspects of the process of utterance," *The relationship of verbal and nonverbal communication*, no. 25, p. 207, 1980.
- [2] M. Leo, G. Medioni, M. Trivedi, T. Kanade, and G. M. Farinella, "Computer vision for assistive technologies," *Computer Vision and Image Understanding*, vol. 154, pp. 1–15, 2017.
- [3] L. P. Berg and J. M. Vance, "Industry use of virtual reality in product design and manufacturing: a survey," *Virtual reality*, vol. 21, no. 1, pp. 1–17, 2017.
- [4] S. S. Rautaray and A. Agrawal, "Vision based hand gesture recognition for human computer interaction: a survey," *Artificial intelligence review*, vol. 43, no. 1, pp. 1–54, 2015.
- [5] C. A. Pickering, K. J. Burnham, and M. J. Richardson, "A research study of hand gesture recognition technologies and applications for human vehicle interaction," in *2007 3rd Institution of Engineering and Technology conference on automotive electronics*. IET, 2007, pp. 1–15.
- [6] F. Parada-Loira, E. González-Agulla, and J. L. Alba-Castro, "Hand gestures to control infotainment equipment in cars," in *IEEE Intelligent Vehicles Symposium*, 2014, pp. 1–6.
- [7] P. Molchanov, X. Yang, S. Gupta, K. Kim, S. Tyree, and J. Kautz, "Online detection and classification of dynamic hand gestures with recurrent 3d convolutional neural network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4207–4215.
- [8] Z. Liu, X. Chai, Z. Liu, and X. Chen, "Continuous gesture recognition with hand-oriented spatiotemporal feature," in *IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2017, pp. 3056–3064.
- [9] G. Zhu, L. Zhang, P. Shen, J. Song, S. Shah, and M. Bennamoun, "Continuous gesture segmentation and recognition using 3dcnn and convolutional lstm," *IEEE Transactions on Multimedia*, 2018.
- [10] P. Narayana, J. R. Beveridge, and B. Draper, "Continuous gesture recognition through selective temporal fusion," in *International Joint Conference on Neural Networks (IJCNN)*. IEEE, 2019.
- [11] O. Köpüklü, A. Gunduz, N. Köse, and G. Rigoll, "Real-time hand gesture detection and classification using convolutional neural networks," in *14th IEEE International Conference on Automatic Face & Gesture Recognition (FG)*, 2019.
- [12] J. Wan, Y. Zhao, S. Zhou, I. Guyon, S. Escalera, and S. Z. Li, "Chalearn looking at people rgb-d isolated and continuous datasets for gesture recognition," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2016.
- [13] Y. Zhang, C. Cao, J. Cheng, and H. Lu, "Egogesture: A new dataset and benchmark for egocentric hand gesture recognition," *IEEE Transactions on Multimedia*, vol. 20, no. 5, pp. 1038–1050, 2018.
- [14] G. Benitez-Garcia, M. Haris, Y. Tsuda, and N. Ukita, "Similar finger gesture recognition using triplet-loss networks," in *Sixteenth IAPR International Conference on Machine Vision Applications (MVA)*. IEEE, 2019, pp. 1–6.
- [15] —, "Finger gesture spotting from long sequences based on multi-stream recurrent neural networks," *Sensors*, vol. 20, no. 2, p. 528, 2020.
- [16] K. Simonyan and A. Zisserman, "Two-stream convolutional networks for action recognition in videos," in *Advances in Neural Information Processing Systems (NeurIPS)*, 2014, pp. 568–576.
- [17] J. Wan, S. Escalera, G. Anbarjafari, H. J. Escalante, X. Baró, I. Guyon, M. Madadi, J. Allik, J. Gorbova, C. Lin *et al.*, "Results and analysis of chalearn lap multi-modal isolated and continuous gesture recognition, and real versus fake expressed emotions challenges," in *IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2017, pp. 3189–3197.
- [18] P. Chao, C.-Y. Kao, Y.-S. Ruan, C.-H. Huang, and Y.-L. Lin, "HarDNet: A Low Memory Traffic Network," in *The IEEE International Conference on Computer Vision (ICCV)*, October 2019.
- [19] G. Huang, Z. Liu, L. van der Maaten, and K. Q. Weinberger, "Densely Connected Convolutional Networks," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.
- [20] D. Tran, L. Bourdev, R. Fergus, L. Torresani, and M. Paluri, "Learning spatiotemporal features with 3d convolutional networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 4489–4497.
- [21] K. Hara, H. Kataoka, and Y. Satoh, "Can spatiotemporal 3d cnns retrace the history of 2d cnns and imagenet?" in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2018, pp. 6546–6555.
- [22] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 770–778.
- [23] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He, "Aggregated residual transformations for deep neural networks," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 1492–1500.
- [24] A. Ranjan and M. J. Black, "Optical flow estimation using a spatial pyramid network," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 4161–4170.
- [25] C. Zimmermann and T. Brox, "Learning to estimate 3d hand pose from single rgb images," in *IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 4903–4911.
- [26] S. Niklaus, "A reimplementation of SPyNet using PyTorch," <https://github.com/sniklaus/pytorch-spynet>, 2018.
- [27] J. Materzynska, G. Berger, I. Bax, and R. Memisevic, "The jester dataset: A large-scale video dataset of human gestures," in *IEEE International Conference on Computer Vision Workshop (ICCVW)*, 2019, pp. 0–0.
- [28] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115, no. 3, pp. 211–252, 2015.
- [29] M. Abavisani, H. R. V. Joze, and V. M. Patel, "Improving the performance of unimodal dynamic hand-gesture recognition with multimodal training," in *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 1165–1174.
- [30] O. Kopuklu, N. Kose, and G. Rigoll, "Motion fused frames: Data level fusion strategy for hand gesture recognition," in *IEEE Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, 2018.