SIGN LANGUAGE RECOGNITION

by

Patrick Raja – 20BCE1058

Adrian V Martin – 20BCE1847

Akash -20BCE1919

A project report submitted to

Dr. Maria Anu

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING

in partial fulfillment for the requirements for the course of

CSE4022 – NATURAL LANGUAGE PROCESSING

in

B.Tech Computer Science and Engineering

VIT®

Vellore Institute of Technology

(Deemed to be University under section 3 of UGC Act, 1956)

APRIL 2023

Abstract

The project aims at building a machine learning model that will be able to classify the various hand gestures used for fingerspelling in sign language. In this user independent model, classification machine learning algorithms are trained using a set of image data and testing is done on a completely different set of data. For the image dataset, depth images are used, which gave better results than some of the previous literatures, owing to the reduced pre-processing time. Various machine learning algorithms are applied on the datasets, including Random Forest Classifier.

Introduction

Communication is very crucial to human beings, as it enables us to express ourselves. We communicate through speech, gestures, body language, reading, writing or through visual aids, speech being one of the most used among them. However, unfortunately, for the speaking and hearing-impaired minority, there is a communication gap. Visual aids, or an interpreter, are used for communicating with them. However, these methods are rather cumbersome and expensive, and can't be used in an emergency. Sign Language chiefly uses manual communication to convey meaning. This involves simultaneously combining hand shapes, orientations and movement of the hands, arms or body to express the speaker's thoughts.

Sign Language consists of fingerspelling, which spells out words character by character, and word level association which involves hand gestures that convey the word meaning. Fingerspelling is a vital tool in sign language, as it enables the communication of names, addresses and other words that do not carry a meaning in word level association. Despite this, fingerspelling is not widely used as it is challenging to understand and difficult to use. Moreover, there is no universal sign language and very few people know it, which makes it an inadequate alternative for communication.

A system for sign language recognition that classifies finger spelling can solve this problem.

Literature Survey

S. No	TITLE	LIST OF AUTHORS	LITERATURE REVIEW	REFERENCE
1.	**Static Hand Gesture Recognition Using Artificial Neural Network**	Trong-Nguyen Nguyen, Huu-Hung Huynh, Jean Meunier	The authors have discussed about a real time system for recognition of American Sign Language (ASL) gestures. The authors then focus on sign language recognition and discuss the various approaches used in this field, including template matching, feature-based approaches, and machine learning techniques. The best achieved results give an accuracy of up to 98%. The authors have identified the key challenges and limitations associated with these approaches. The authors provide a detailed overview of the features and limitations of each system and highlight the need for real-time, accurate, and context-aware sign language recognition systems.	https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a240159a3e11010ef51ab767da234085fc06e90b
2.	**Computer Vision Based Hand Gesture Interfaces for Human-Computer Interaction**	Sören Lenman, Lars Bretzner, Björn Thuresson	This article explains about building a first prototype for exploring the use of pie- and marking menus in gesture-based interaction. The authors have chosen a viewbased representation of the hand, including both color and shape cues. The system tracks and recognizes the hand poses based on a combination of multi-scale color feature detection, view-based hierarchical hand models and particle filtering. The hand poses, or hand states, are represented in terms of hierarchies of color image features at different scales, with qualitative inter-relations in terms of scale, position and orientation. These hierarchical models capture the coarse shape of the hand poses. In each image, detection of multi-scale colour features is performed. The hand states are then simultaneously detected and tracked using particle filtering, with an extension of	https://cid.nada.kth.se/pdf/CID-172.pdf

			layered sampling referred to as hierarchical layered sampling.	
3.	**Hand Gesture Recognition using Deep Learning for Sign Language Interpretation**	N. R. Das, B. R. Shadap, and D. K. Bhattacharyya	The authors have discussed the advantages of deep learning-based approaches for hand gesture recognition, such as their ability to automatically learn features from data, handle variations in hand gestures, and improve recognition accuracy. They also provide a brief overview of the various deep learning architectures used in hand gesture recognition, such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and their variants. The authors have discussed about the need for a system to identify the hand sign gustures in real time. The authors highlight the limitations of the machine learning approaches, including the need for a large dataset, high computational complexity, and poor performance in noisy environments and suggests a alternative deep-learning model.	https://ieeexplore.ieee.org/document/8351856
4.	**Android based American Sign Language Recognition System with Skin Segmentation and SVM**	Sakshi Lahoti, Shaily Kayal, Sakshi Kumbhare, Ishani Suradkar, Vikul Pawar	In this article, The authors speaks about building a mobile app that can recognizes sign language. This article plans on creating a model using three steps namely, Hand gesture capture and skin, feature extraction, classification using SVM, over 500 images are used for training, The project uses skin segmentation with a black background so that akin segmentation and edge detection becomes easy and reduce the storage space and load time. The model displays the output on the screen when a input hand gesture is shown in the camera. Support vector machine (SVM) is a supervised learning model which is used in classification of model. In this article they have used HOG (histogram of oriented gradient) which acts as a human/feature descriptor for this purpose. The final dataset is sent to	https://ieeexplore.ieee.org/abstract/document/8493838/references#references

			cloud for further requirement. The accuracy aquired by the android app is about 89.54 percentage. The author says that the accuracy varies with the complex background. This can be very useful because mobile phone is one of the widely used electronic device.	
5.	**Sign Language Recognition**	Satwik Ram Kodandaram, N. Pavan Kumar, Sunil Gl	This research paper deals with recognizing the hand gestures acquisition and continues till text or speech is generated for corresponding hand gestures. Here hand gestures for sign language can be classified as static and dynamic. However, static hand gesture recognition is simpler than dynamic hand gesture recognition, but both recognition is important to the human community. The authors use Deep Learning Computer Vision to recognize the hand gestures by building Deep Neural Network architectures (Convolution Neural Network Architectures) where the model will learn to recognize the hand gestures images over an epoch.	(PDF) Sign Language Recognition (researchgate.net)
6.	**Sign and Human Action Detection Using Deep Learning**	Shivanarayna Dhulipala, Festus Fatai Adedoyin, Alessandro Bruno	The authors of this study aimed to develop an efficient deep learning model that can be used to predict British sign language in an attempt to narrow this communication gap between speech-impaired and non-speech-impaired people in the community. Two models were developed in this research, CNN and LSTM, and their performance was evaluated using a multi-class confusion matrix. The CNN model emerged with the highest performance, attaining training and testing accuracies of 98.8% and 97.4%, respectively. In addition, the model achieved average weighted precession and recall of 97% and 96%, respectively. On the other hand, the LSTM model's performance was quite poor, with the maximum training and testing	Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images | IEEE Conference Publication | IEEE Xplore

			performance accuracies achieved being 49.4% and 48.7%, respectively. The research concluded that the CNN model was the best for recognizing and determining British sign language.	
7.	**Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images**	Aditya Das, Shantanu Gawde, Khyati Suratwala, Dhananjay Kalbande	The authors of the paper presents results obtained by retraining and testing this sign language gestures dataset on a convolutional neural network model using Inception v3. The model consists of multiple convolution filter inputs that are processed on the same input. The validation accuracy obtained was above 90% This paper also reviews the various attempts that have been made at sign language detection using machine learning and depth data of images. It takes stock of the various challenges posed in tackling such a problem, and outlines future scope as well.	Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images | IEEE Conference Publication | IEEE Xplore
8.	**Faster Convergence and Reduction of Overfitting in Numerical Hand Sign Recognition Using DCNN**	Abdul Kawsar Tushar, Akm Ashiquzzaman , and Md. Rashedul Islam	In this article, the authors propose several modifications to the traditional DCNN architecture. They introduce a dropout layer to reduce overfitting, a batch normalization layer to reduce internal covariate shift and achieve faster convergence, and a global average pooling layer to reduce the number of parameters. The authors report that the modified DCNN achieves higher accuracy and faster convergence compared to the traditional DCNN architecture. They also demonstrate that the modified DCNN outperforms other state-of-the-art methods for numerical hand sign recognition.	https://ieeexplore.ieee.org/abstract/document/8289040
9.	**Hand gesture recognition using a real-time tracking method and hidden Markov models**	Feng-Sheng Chen, Chih-Ming Fu, Chung-Lin Huang	In order to distinguish continuous gestures against stationary backgrounds, the authors of this study develop a hand gesture recognition system. The system consists of four modules: gesture recognition, feature extraction, hidden Markov model training, and real-time hand tracking and extraction. They first employ a real-	https://www.sciencedirect.com/science/article/abs/pii/S0262885603000702

			time hand tracking and extraction technique to track the moving hand and extract the hand region, after which they characterise the spatial data using a Fourier descriptor and the temporal characteristics using a motion analysis. The authors combine the spatial and temporal features of the input image sequence as their feature vector. They then use HMMs to recognise the input gesture after extracting the feature vectors. The gesture that has to be recognised is evaluated separately by each HMM. The relevant gesture is shown by the model with the highest score. The validation accuracy obtained was above 90%.	
10.	**Hand gesture recognition using a neural network shape fitting technique**	E. Stergiopoulou, N. Papamarkos	In this paper proposes a new technique for hand gesture recognition which is based on hand gesture features and on a neural network shape fitting procedure. Firstly, the hand region is isolated by using a skin color filtering procedure in the YCbCr color space. To obtain noiseless segmented images regardless to the variation of the skin color and the lighting conditions. The stage that concerns the fitting of the hand's shape as well as the stage of finger features extraction is the innovative and powerful Self-Growing and Self Organized Neural Gas network which approximate the hand's morphology. As a result, the extracted finger features are well discriminated and thus they conduce to a successful recognition. It is found from the experiments that the recognition rate is very promising and approaches 90.45%. It is worth to underline that the key characteristic of the proposed hand gesture recognition technique is the use of the SGONG neural network. The reason is twofold; SGONG is able to describe very effectively the shape of the hand, and thus allows the extraction of robust and effective	https://www.sciencedirect.com/science/article/abs/pii/S0952197609000694

			features, and moreover it achieves it by converging faster than other networks.	

Libraries and Models Used:

- **OpenCV** is a popular Python library used for computer vision tasks, including image and video processing, object detection and recognition, feature extraction, and camera calibration.

- **Mediapipe** uses a machine learning model based on convolutional neural networks (CNNs) to estimate the 3D coordinates of 21 hand landmarks and visualizing it.

- **Pickle** used to serialize and deserialize a wide range of Python objects, including integers, floating-point numbers, strings, lists, dictionaries, sets, and complex data structures containing these basic types.

- **Random Forest Model** builds multiple decision trees and combines their predictions to produce a more accurate and stable prediction.

- **Numpy** provides a set of powerful tools for working with multi-dimensional arrays and matrix.

Modules:

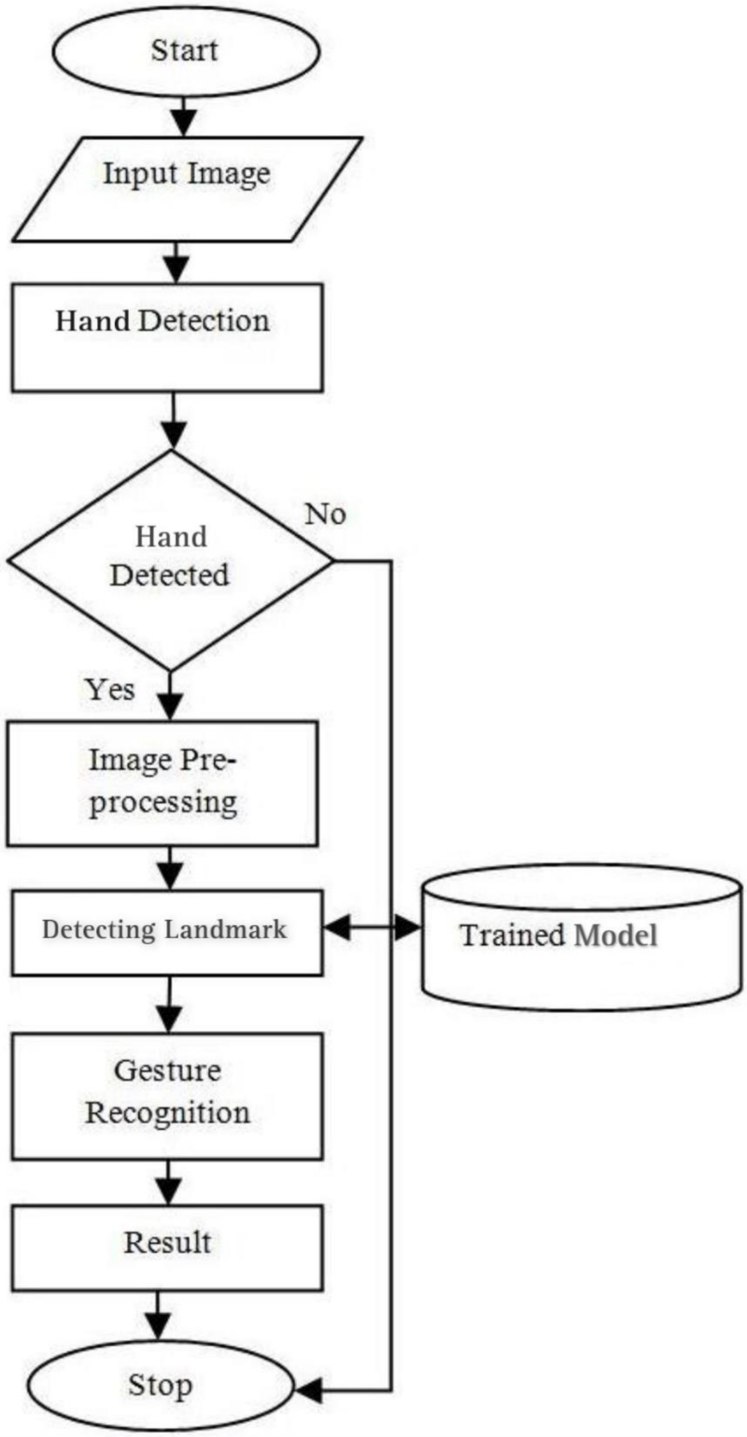
- Collecting image and labelling the folder.
- Using mediapipe for collecting landmarks co-ordinates from the images
- Collecting the landmarks co-ordinates from each images in the dataset and storing it in a pickle file.
- Training the extracted values using any machine learning model and recording the accuracy.
- Using different machine learning model for best accuracy.
- Storing the model.
- Using the model to predict real time sign languages and displaying the prediction.

REQUIREMENTS:

- IDE: Pycharm
- Library
  1. Os
  2. Cv2
  3. Pickle
  4. Mediapipe
  5. Random Forest Classifier from sklearn.ensemble
  6. Train_test_split from sklearn.model_selection
  7. Accuracy_score from sklearn.metrics
  8. Numpy

Experimental Setup

Architecture diagram:



Procedure:

Collecting images:

Create a folder named data for storing the train data(images). Mention its path in a variable as string. If the folder is not created the program will create a folder. Specify the number of signs you are going to train in the number_of_classes variable and the number of images collected in the dataset_size variable. Using cv2, The frame of the camera footage is displayed. Press “Q” for capturing images continuously for each class. Repeat the same for total number of classes specified.

Creating dataset:

Using mediapipe library, we can obtain the landmarks of the hands in the images. Each images in the folders is changes from BGR format to RGB format. The images are processed to find the landmarks. The results are stored in an array. By subtracting the minimum x and y values from each hand landmark point's coordinates, the code effectively normalizes the hand landmark data with respect to the top-left corner of the hand bounding box. This normalization is useful for various tasks such as gesture recognition, as it removes the dependency of hand landmark data on the absolute position and size of the hand in the video frame. The results such as the co-ordinates and the directory name which is represented as labels is stored in a pickle file.

Training the dataset:

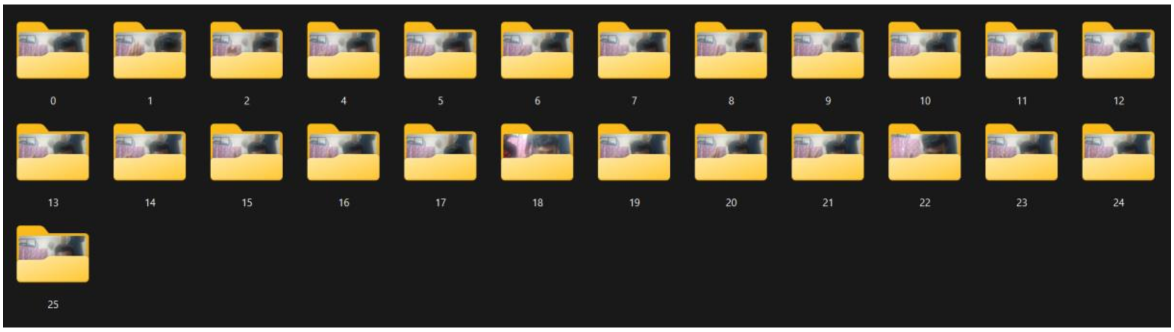
Using the data and label values from the pickle file. The data is split into two types training and testing data. The training data consists of 80% of the dataset (images) and test data consists of 20% of the dataset. An ensemble model is built using random forest classifier and trained using the training data. The prediction accuracy is found to be 100%. The model is stored for real time prediction.

Inference Classifier:

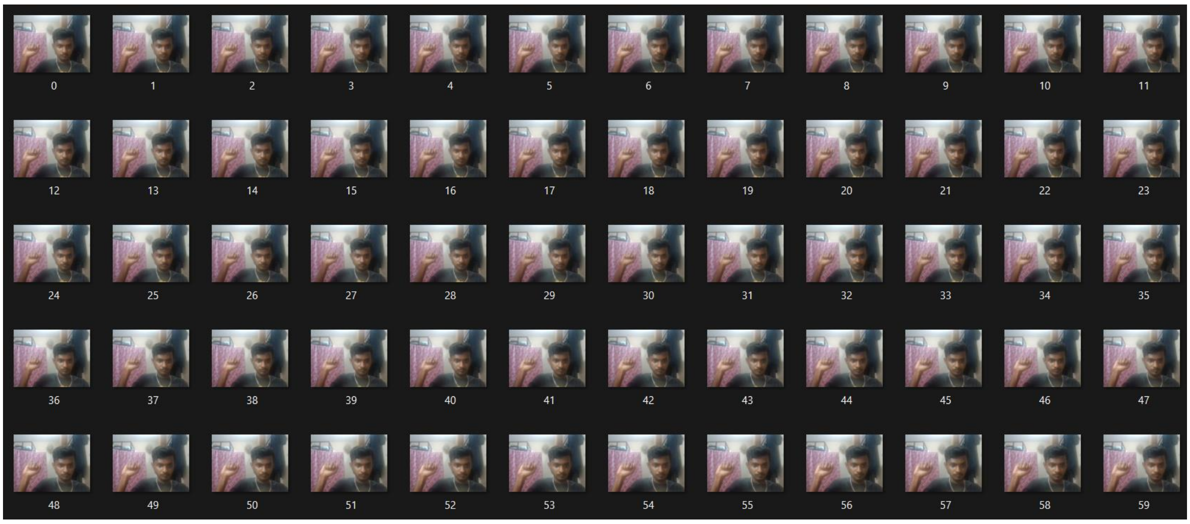
In inference classifier, a real time image is taken from the camera and the image is changed from BGR to RGB format. The landmarks are stored and using the prediction model the result is predicted and stored as a variable. Using cv2 the predicted value is displayed near the hand in the camera.

Output Screenshots :

Collecting images:

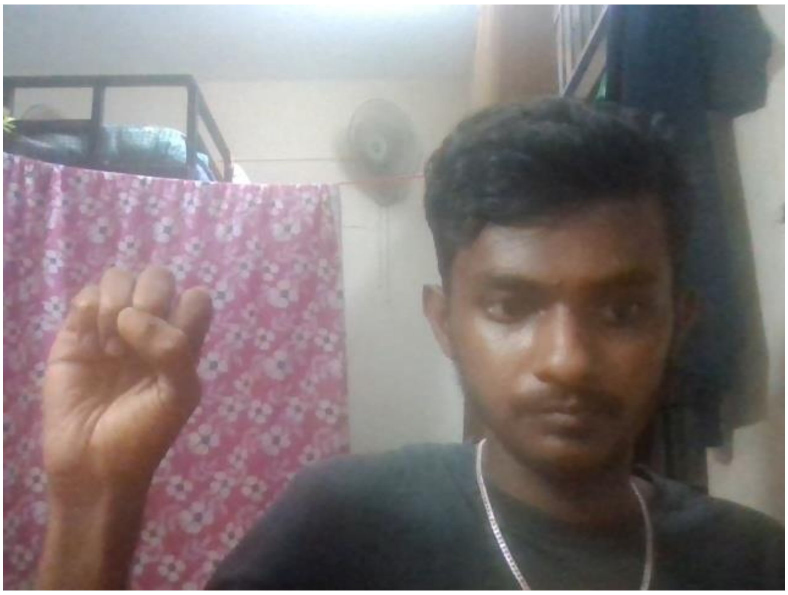


100 images are stored:



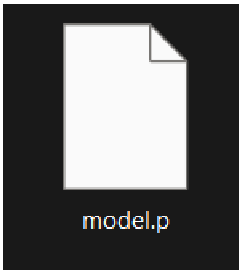
Single image:

An training image for the alphabet ‘E’.

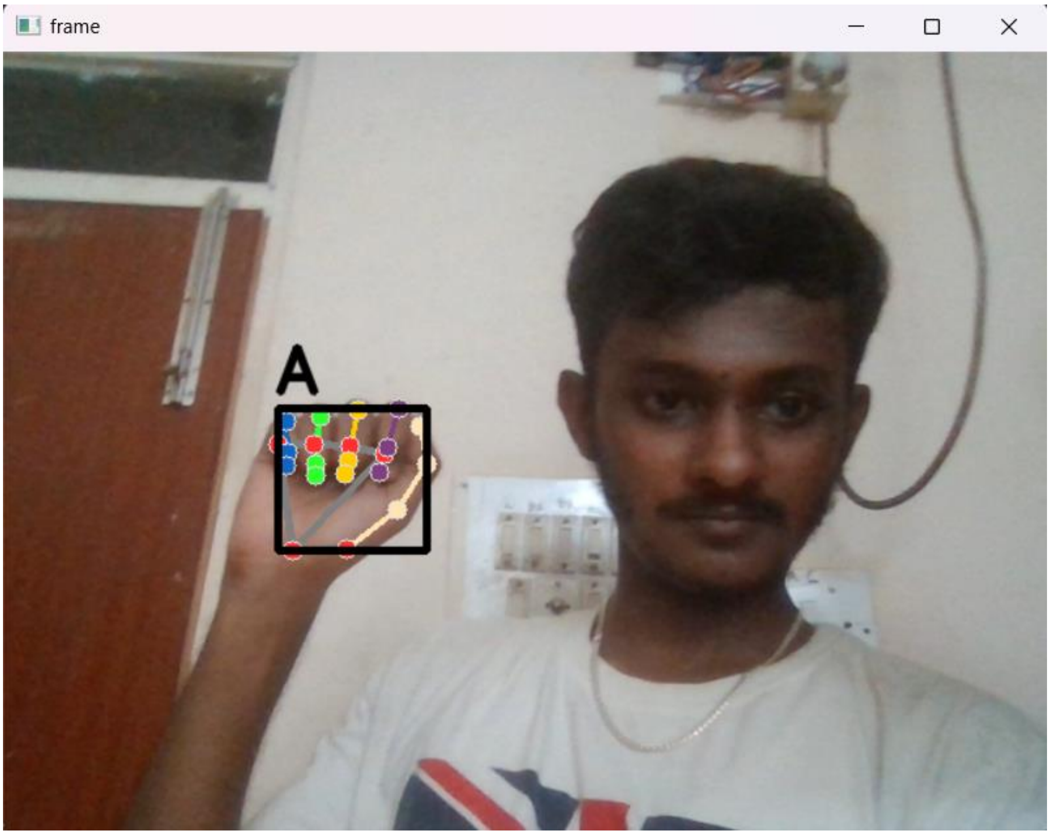


Storing training dataset in the pickle file:

Creating a model using Random forest and saving it:



Predicting real time sign language:



Conclusion:

This project can increase the interaction between the people who don't know sign language with people who are unable to ear or talk. Using this project any one can train their own gesture and share it with others, so that any confidential matter can be shared without others understanding it.

By using the ensemble Random Forest model on this extracted data, we demonstrated high levels of accuracy in detecting and classifying sign language gestures.

Reference:

- https://citeseerx.ist.psu.edu/document?repid=rep1&type=pdf&doi=a240159a3e11010ef51ab767da234085fc06e90b
- https://cid.nada.kth.se/pdf/CID-172.pdf
- Sign Language Recognition Using Deep Learning on Custom Processed Static Gesture Images | IEEE Conference Publication | IEEE Xplore

Appendix:

COLLECT_IMGS.PY:

```
import os
import cv2


DATA_DIR = './data'
if not os.path.exists(DATA_DIR):
    os.makedirs(DATA_DIR)

number_of_classes = 26
dataset_size = 100

cap = cv2.VideoCapture(0)
for j in range(number_of_classes):
    if not os.path.exists(os.path.join(DATA_DIR, str(j))):
        os.makedirs(os.path.join(DATA_DIR, str(j)))

    print('Collecting data for class {}'.format(j))

    done = False
    while True:
        ret, frame = cap.read()
        cv2.putText(frame, 'Ready? Press "Q" ! :)', (100, 50),
cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 255, 0), 3,cv2.LINE_AA)
        cv2.imshow('frame', frame)
        if cv2.waitKey(25) == ord('q'):
            break

    counter = 0
    while counter < dataset_size:
        ret, frame = cap.read()
        cv2.imshow('frame', frame)
        cv2.waitKey(25)
        cv2.imwrite(os.path.join(DATA_DIR, str(j),
'{}.jpg'.format(counter)), frame)

        counter += 1

cap.release()
cv2.destroyAllWindows()
```

CREATE_DATASET.PY:

```
import os
import pickle

import mediapipe as mp
import cv2


mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True,
min_detection_confidence=0.3)
```

```python
DATA_DIR = './data'

data = []
labels = []
for dir_ in os.listdir(DATA_DIR):
    for img_path in os.listdir(os.path.join(DATA_DIR, dir_)):
        data_aux = []

        x_ = []
        y_ = []

        img = cv2.imread(os.path.join(DATA_DIR, dir_, img_path))
        img_rgb = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)

        results = hands.process(img_rgb)
        if results.multi_hand_landmarks:
            for hand_landmarks in results.multi_hand_landmarks:
                for i in range(len(hand_landmarks.landmark)):
                    x = hand_landmarks.landmark[i].x
                    y = hand_landmarks.landmark[i].y

                    x_.append(x)
                    y_.append(y)

                for i in range(len(hand_landmarks.landmark)):
                    x = hand_landmarks.landmark[i].x
                    y = hand_landmarks.landmark[i].y
                    data_aux.append(x - min(x_))
                    data_aux.append(y - min(y_))

            data.append(data_aux)
            labels.append(dir_)

f = open('data.pickle', 'wb')
pickle.dump({'data': data, 'labels': labels}, f)
f.close()
```

TRAIN_CLASSIFIER.PY:

```python
import pickle

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np


data_dict = pickle.load(open('./data.pickle', 'rb'))

data = np.asarray(data_dict['data'])
labels = np.asarray(data_dict['labels'])

x_train, x_test, y_train, y_test = train_test_split(data, labels,
test_size=0.2, shuffle=True)

model = RandomForestClassifier()

model.fit(x_train, y_train)
```

```
y_predict = model.predict(x_test)

score = accuracy_score(y_predict, y_test)

print('{}% of samples were classified correctly !'.format(score * 100))

f = open('model.p', 'wb')
pickle.dump({'model': model}, f)
f.close()
```

INFERENCE_CLASSIFIER.PY:

```
import pickle

import cv2
import mediapipe as mp
import numpy as np

model_dict = pickle.load(open('model.p', 'rb'))
model = model_dict['model']

cap = cv2.VideoCapture(0)

mp_hands = mp.solutions.hands
mp_drawing = mp.solutions.drawing_utils
mp_drawing_styles = mp.solutions.drawing_styles

hands = mp_hands.Hands(static_image_mode=True,
min_detection_confidence=0.3)

labels_dict = {0: 'A', 1: 'B', 2: 'C',3: 'D',4: 'E',5: 'F',6: 'G',7: 'H',8:
'I',9: 'J',10: 'K',11: 'L',12: 'M',13: 'N',14: 'O',15:
'P',16:'R',17:'Q',18:'S',19:'T',20:'U',21:'V',22:'W',23:'X',24:'Y',25:'Z'}
while True:

    data_aux = []
    x_ = []
    y_ = []

    ret, frame = cap.read()

    H, W, _ = frame.shape

    frame_rgb = cv2.cvtColor(frame, cv2.COLOR_BGR2RGB)

    results = hands.process(frame_rgb)
    if results.multi_hand_landmarks:
        for hand_landmarks in results.multi_hand_landmarks:
            mp_drawing.draw_landmarks(
                frame,  # image to draw
                hand_landmarks,  # model output
                mp_hands.HAND_CONNECTIONS,  # hand connections
                mp_drawing_styles.get_default_hand_landmarks_style(),
                mp_drawing_styles.get_default_hand_connections_style())

        for hand_landmarks in results.multi_hand_landmarks:
            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
```

```
                x_.append(x)
                y_.append(y)

            for i in range(len(hand_landmarks.landmark)):
                x = hand_landmarks.landmark[i].x
                y = hand_landmarks.landmark[i].y
                data_aux.append(x - min(x_))
                data_aux.append(y - min(y_))

        x1 = int(min(x_) * W)
        y1 = int(min(y_) * H)

        x2 = int(max(x_) * W)
        y2 = int(max(y_) * H)

        prediction = model.predict([np.asarray(data_aux)])

        predicted_character = labels_dict[int(prediction[0])]

        cv2.rectangle(frame, (x1, y1), (x2, y2), (0, 0, 0), 4)
        cv2.putText(frame, predicted_character, (x1, y1 - 10),
cv2.FONT_HERSHEY_SIMPLEX, 1.3, (0, 0, 0), 3,
                    cv2.LINE_AA)

    cv2.imshow('frame', frame)
    cv2.waitKey(1)


cap.release()
cv2.destroyAllWindows()
```