# SIGN LANGUAGE DETECTION

*Submitted by*

**GUNTI MEGHNATH [RA2011026010219]**
**SASHANK DONAVALLI[RA2011026010234]**
**HARSHITHA KAMBHAM [RA2011026010235]**

*Under the Guidance of*

**Dr. U.SAKTHI**

**Assistant Professor, Department of Computational Intelligence**

*In partial satisfaction of the requirements for the degree of*

**BACHELORS OF TECHNOLOGY**
**in**
**COMPUTER SCIENCE ENGINEERING**

**with specialization in Artificial Intelligence & Machine Learning**


**SCHOOL OF COMPUTING**

**COLLEGE OF ENGINEERING AND TECHNOLOGY**

**SRM INSTITUTE OF SCIENCE AND TECHNOLOGY**

**KATTANKULATHUR - 603203**

**May 2023**

SRM INSTITUTION OF SCIENCE AND TECHNOLOGY
KATTANKULATHUR-603203

# BONAFIDE CERTIFICATE

Certified that this Course Project Report titled **"SIGN LANGUAGE DETECTION"** is the bonafide work done by **GUNTI MEGHNATH [RA2011026010219], SASHANK DONAVALLI [RA2011026010234] and HARSHITHA KAMBHAM [RA2011026010235]**who carried out under my supervision. Certified further, that to the best of my knowledge the work reported herein does not form part of any other work.


**SIGNATURE**
Faculty In-Charge
**Dr.U.Sakthi**
Assistant Professor
Department of Networking and Communications
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

**HEAD OF THE DEPARTMENT**
**Dr. R Annie Uthra**
Professor and Head
Department of Computational Intelligence,
SRM Institute of Science and Technology
Kattankulathur Campus, Chennai

# ABSTRACT

Hand gesture is one of the methods used in sign language for non-verbal communication. It is most commonly used by deaf & dumb people who have hearing or speech problems to communicate among themselves or with normal people. Various sign language systems have been developed by many makers around the world but they are neither flexible nor cost-effective for the end users. Hence in this paper introduced software which presents a system prototype that is able to automatically recognize the alphabets used in sign language to help deaf and dumb people to communicate more effectively with each other or normal people. Pattern recognition and Gesture recognition are the developing fields of research. Being a significant part in nonverbal communication hand gestures are playing key role in our daily life.

Sign Language has long been the main form of communication in the community of people with special needs. It uses a visual modality for information exchange in addition to syntax and vocabulary, much like any other language. The issue occurs when dumb or deaf persons attempt to communicate with others using this sign language syntax.
This way of communication is only available to the family of the aforementioned specially helped person because it is quite recognizable to anyone outside of their society.

# TABLE OF CONTENTS

# ABBREVIATIONS

| | |
|---|---|
| **CNN** | Convolutional neural network |
| **OpenCV** | Computer Vision |
| **DHT** | Distributed hash table |
| **IR** | Infra-red |
| **HCI** | Universal Human Computer Interface |
| **IDE** | Integrated Development Environment |

# INTRODUCTION

It can be quite difficult to communicate with people who have hearing loss. Due to the hand gestures used by Deaf and Mute people to communicate, it can be challenging for people outside of their society to understand the signs they use to communicate. Systems that can identify various indications and provide information to the public are thus necessary.

A sign detector is used to identify hand and other gestures that are utilized in sign language. Based on these precise motions, the detector then shows signals for the persons who require particular assistance. In this context, machine learning and computer vision ideas can be applied.

The objective is to familiarize the computer with human speech recognition and develop an intuitive human-computer interface (HCI). This method includes several steps, such as teaching your computer to recognize the alphabets used in sign language.

# LITERATURE SURVEY

## 2.1 American Sign Language Recognition

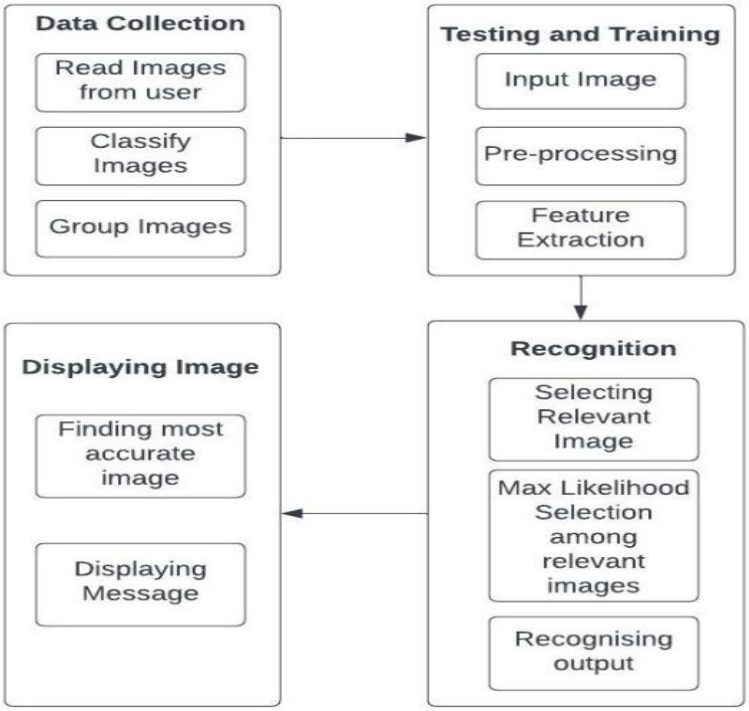
American sign language, and sign languages generally, consist of manual and nonmanual signing gestures. Hands- and arm-only gestures are used for manual signing. Non-manual signing includes wider torso and head movements in addition to face expressions. The majority of a sentence's lexical meaning may often be expressed via manual signals. The non-manual features of the sign then provide further information. These specifics could include verb tenses, tone, or the force of the action. Although it would be ideal for an ASLR system to support both manual and non-manual gestures, doing so is not straightforward. The shape, position, placement, and movement of the hand, including the palm and fingers, are the fundamental elements of a manual sign. Perlmutter believes that signs are made up of two forms of segmentation: position and movement. There are further secondary movements inside these segments that might be viewed as "internal movements" of the fingers with respect to the hand. An ASLR-5 system must be able to model both the static and moving parts of signs in order to perform accurate classification. Additionally, there is movement between the various signs that is not a part of a sign and does not truly transmit any meaning when signs are combined to form sentences.

## 2.2 Existing systems:

Glove-based technique of the existing system: In this method, the signer is required to wear a hardware glove while their hand movements are being recorded. The glove's movement will be used to generate a message.

Vision-based method: It can be further divided into static and dynamic recognition. Static recognition focuses on the diagnosis of static gestures (2-dimensional images), whereas dynamic recognition focuses on the live recording of movements in real-time. This involves using a camera to capture movement.

# SYSTEM ARCHITECTURE AND DESIGN

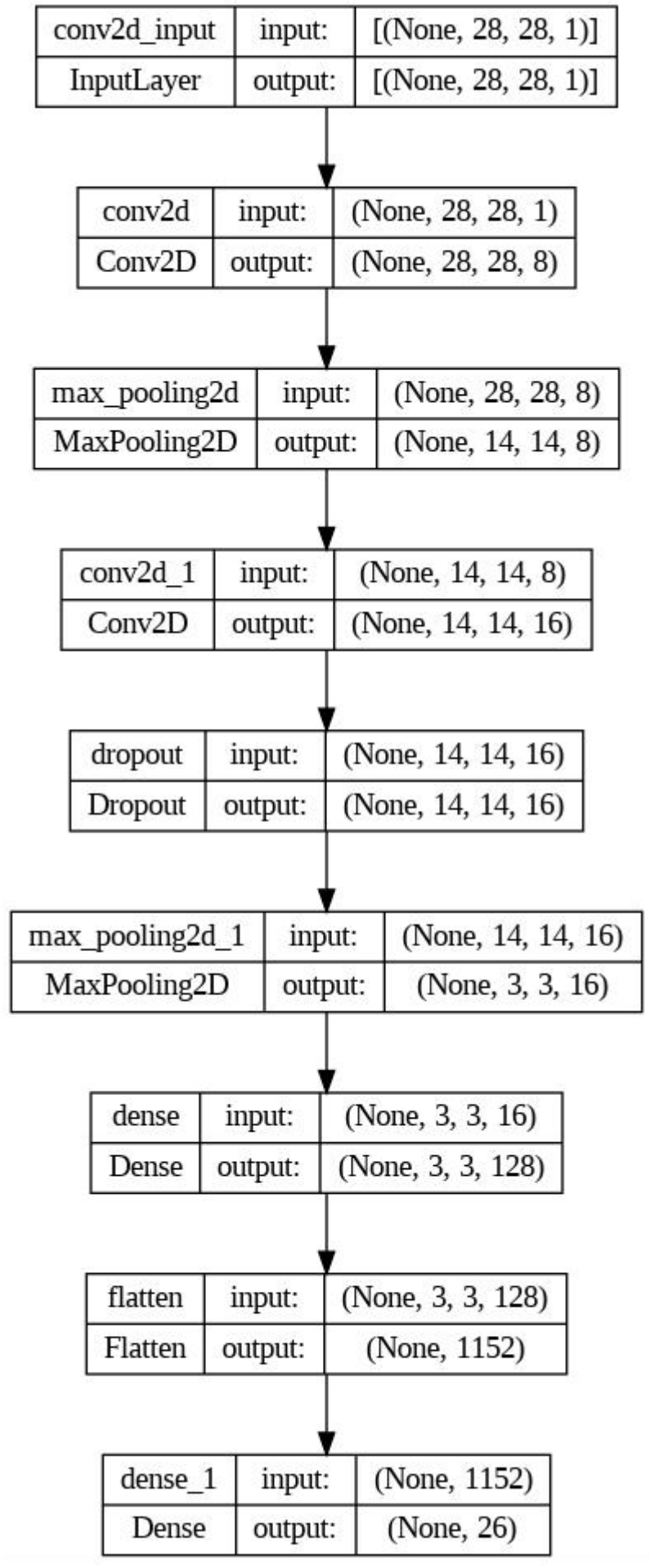



Phase 1: Data Collection: A model is being developed, and it is being fed a series of photographs. This stage is important for subsequent symbol-based model training.

Phase 2: Training and Testing: In this step, a set of model inputs are used, and a certain outcome is anticipated. The correctness of the model is being determined based on the results.

Phase 3: Recognition of output: An image through OpenCV is provided to the model as input during this phase. It generates a message and matches each image with a certain output based on the photos being trained into the model.

Phase 4: Output: The output is displayed by the user after recognizing the alphabet.




**The CNN Architecture**

# METHODOLOGY

The methodology for building an CNN for Sign language detection for alphabets classification using python typically involves the following steps:

Data Preparation: The first step is to prepare the data for training the CNN. This involves downloading the SignlanguageMNISTdataset from Kaggle, which consists of 5456 training images and 5278 testing images. The dataset is typically split into training and testing sets, with the training set used for training the CNN and the testing set used for evaluating its performance. The images are typically preprocessed by normalizing the pixel values to be between 0 and 1, and reshaped to a 2D array.

Model Definition: The next step is to define the CNN model. This involves selecting the type of layers to use, such as dense layers, convolutional layers, or pooling layers, and specifying the number of neurons and activation functions for each layer. The model can be defined using a high-level API like Keras, which provides a simple interface for building CNNs.

Model Training: Once the model is defined, it can be trained on the training set using an optimization algorithm like Stochastic Gradient Descent (SGD). During training, the model iteratively adjusts its parameters to minimize the loss function, which measures the difference between the predicted and actual labels. The training process can be monitored using metrics like accuracy and loss, and early stopping can be used to prevent overfitting

Model Evaluation: Once the model is trained, it can be evaluated on the testing set to measure its performance. This involves using metrics like accuracy, precision, recall, to assess how well the model is able to classify the images.

Hyperparameter Tuning: The performance of the CNN model can be improved by tuning the hyperparameters, such as the learning rate, batch size, and number of layers. This involves trying different combinations of hyperparameters and evaluating the performance on a validation set.

Deployment: Finally, once the model is trained and tuned, it can be deployed for use in real-world applications. This typically involves saving the model weights and architecture, and loading them into a production environment where new images can be classified using the trained CNN. Overall, the methodology for building an CNN for Signlanguage MNIST classification involves data preparation, model definition, model training, model evaluation, hyperparameter tuning, and deployment. By following this methodology, it is possible to build an accurate and reliable ANN for classifying images in the SignlanguageMNIST dataset.

# CODING AND TESTING

```
import keras
import numpy as np
import pandas as pd
import cv2
from matplotlib import pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D, Dense,Flatten, Dropout
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.optimizers import SGD
train = pd.read_csv(' /content/sign_mnist_train.csv')
download = drive.CreateFile({'id': '1q_Zwlu3RncjKq1YpiVtkiMPxIIueGRYB'})
download.GetContentFile('test.csv')
test = pd.read_csv(' /content/sign_mnist_test.csv')
y_train = train['label'].values
y_test = test['label'].values
X_train = train.drop(['label'],axis=1)
X_test = test.drop(['label'], axis=1)
X_train = np.array(X_train.iloc[:,:])
X_train = np.array([np.reshape(i, (28,28)) for i in X_train])
X_test = np.array(X_test.iloc[:,:])
X_test = np.array([np.reshape(i, (28,28)) for i in X_test])
num_classes = 26
y_train = np.array(y_train).reshape(-1)
y_test = np.array(y_test).reshape(-1)
y_train = np.eye(num_classes)[y_train]
y_test = np.eye(num_classes)[y_test]
```

```
X_train = X_train.reshape((27455, 28, 28, 1))

X_test = X_test.reshape((7172, 28, 28, 1))

classifier = Sequential()

classifier.add(Conv2D(filters=8,
kernel_size=(3,3),strides=(1,1),padding='same',input_shape=(28,28,1),activation='relu',
data_format='channels_last'))

classifier.add(MaxPooling2D(pool_size=(2,2)))

classifier.add(Conv2D(filters=16, kernel_size=(3,3),strides=(1,1),padding='same',activation='relu'))

classifier.add(Dropout(0.5))

classifier.add(MaxPooling2D(pool_size=(4,4)))

classifier.add(Dense(128, activation='relu'))

classifier.add(Flatten())

classifier.add(Dense(26, activation='softmax'))

classifier.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])

classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])

classifier.fit(X_train, y_train, epochs=50, batch_size=100)

accuracy = classifier.evaluate(x=X_test,y=y_test,batch_size=32)

print("Accuracy: ",accuracy[1])

classifier.summary()

from keras.utils.vis_utils import plot_model

plot_model(classifier, to_file='model_plot.png', show_shapes=True, show_layer_names=True)

!apt install graphviz

!pip install pydot pydot-ng

!echo "Double check with Python 3"

!python -c "import pydot"

plot_model(classifier, show_shapes=True, show_layer_names=True, to_file='model.png')

from IPython.display import Image

Image(retina=True, filename='model.png')
```

# SCREENSHOTS AND RESULTS

+ Code + Text | Copy to Drive | RAM Disk

```
import keras
import numpy as np
import pandas as pd
import cv2
from matplotlib import pyplot as plt
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D, Dense,Flatten, Dropout
from keras.datasets import mnist
import matplotlib.pyplot as plt
from keras.utils import np_utils
from keras.optimizers import SGD
```

```
[73] !pip install PyDrive
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-wheels/public/simple/
Requirement already satisfied: PyDrive in /usr/local/lib/python3.10/dist-packages (1.3.1)
Requirement already satisfied: PyYAML>=3.0 in /usr/local/lib/python3.10/dist-packages (from PyDrive) (6.0)
Requirement already satisfied: google-api-python-client>=1.2 in /usr/local/lib/python3.10/dist-packages (from Py
Requirement already satisfied: oauth2client>=4.0.0 in /usr/local/lib/python3.10/dist-packages (from PyDrive) (4.
Requirement already satisfied: google-auth<3.0.0dev,>=1.19.0 in /usr/local/lib/python3.10/dist-packages (from go
Requirement already satisfied: uritemplate<5,>=3.0.1 in /usr/local/lib/python3.10/dist-packages (from google-api
Requirement already satisfied: httplib2<1dev,>=0.15.0 in /usr/local/lib/python3.10/dist-packages (from google-ap
Requirement already satisfied: google-auth-httplib2>=0.1.0 in /usr/local/lib/python3.10/dist-packages (from goog
```

```
[67] y_train = train['label'].values
     y_test = test['label'].values

     X_train = train.drop(['label'],axis=1)
     X_test = test.drop(['label'], axis=1)

     X_train = np.array(X_train.iloc[:,:])
     X_train = np.array([np.reshape(i, (28,28)) for i in X_train])

     X_test = np.array(X_test.iloc[:,:])
     X_test = np.array([np.reshape(i, (28,28)) for i in X_test])

     num_classes = 26
     y_train = np.array(y_train).reshape(-1)
     y_test = np.array(y_test).reshape(-1)

     y_train = np.eye(num_classes)[y_train]
     y_test = np.eye(num_classes)[y_test]
```

```
[68] X_train = X_train.reshape((27455, 28, 28, 1))
     X_test = X_test.reshape((7172, 28, 28, 1))
```

```
[69]
     classifier = Sequential()
     classifier.add(Conv2D(filters=8, kernel_size=(3,3),strides=(1,1),padding='same',input_shape=(28,28,1),activati
     classifier.add(MaxPooling2D(pool_size=(2,2)))
     classifier.add(Conv2D(filters=16, kernel_size=(3,3),strides=(1,1),padding='same',activation='relu'))
     classifier.add(Dropout(0.5))
     classifier.add(MaxPooling2D(pool_size=(4,4)))
     classifier.add(Dense(128, activation='relu'))
     classifier.add(Flatten())
     classifier.add(Dense(26, activation='softmax'))
     classifier.compile(optimizer='SGD', loss='categorical_crossentropy', metrics=['accuracy'])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/framework/op_def_library.py:263
Instructions for updating:
Colocations handled automatically by placer.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/tensorflow_backend.py:3445: calling
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - keep_prob`.
```

```
classifier.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
classifier.fit(X_train, y_train, epochs=50, batch_size=100)
```

```
Epoch 1/50
275/275 [==============================] - 3s 6ms/step - loss: 4.3020 - accuracy: 0.3644
Epoch 2/50
275/275 [==============================] - 2s 6ms/step - loss: 0.7851 - accuracy: 0.7337
Epoch 3/50
275/275 [==============================] - 1s 5ms/step - loss: 0.4960 - accuracy: 0.8275
Epoch 4/50
275/275 [==============================] - 1s 4ms/step - loss: 0.3644 - accuracy: 0.8713
Epoch 5/50
275/275 [==============================] - 1s 5ms/step - loss: 0.2842 - accuracy: 0.8989
Epoch 6/50
275/275 [==============================] - 1s 4ms/step - loss: 0.2424 - accuracy: 0.9153
Epoch 7/50
275/275 [==============================] - 1s 4ms/step - loss: 0.2014 - accuracy: 0.9303
Epoch 8/50
275/275 [==============================] - 1s 5ms/step - loss: 0.1885 - accuracy: 0.9333
```

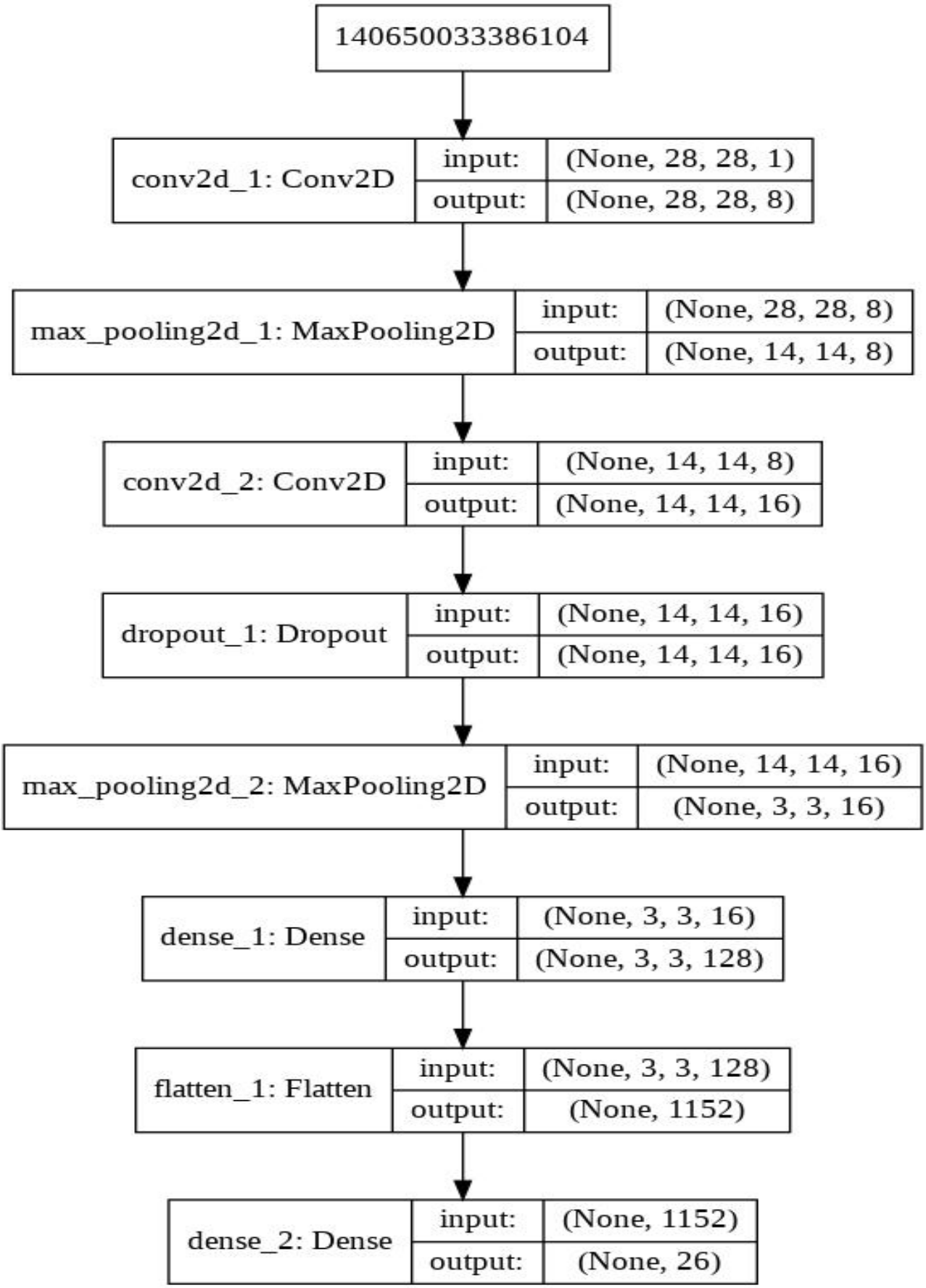
```
accuracy = classifier.evaluate(x=X_test,y=y_test,batch_size=32)
print("Accuracy: ",accuracy[1])
```

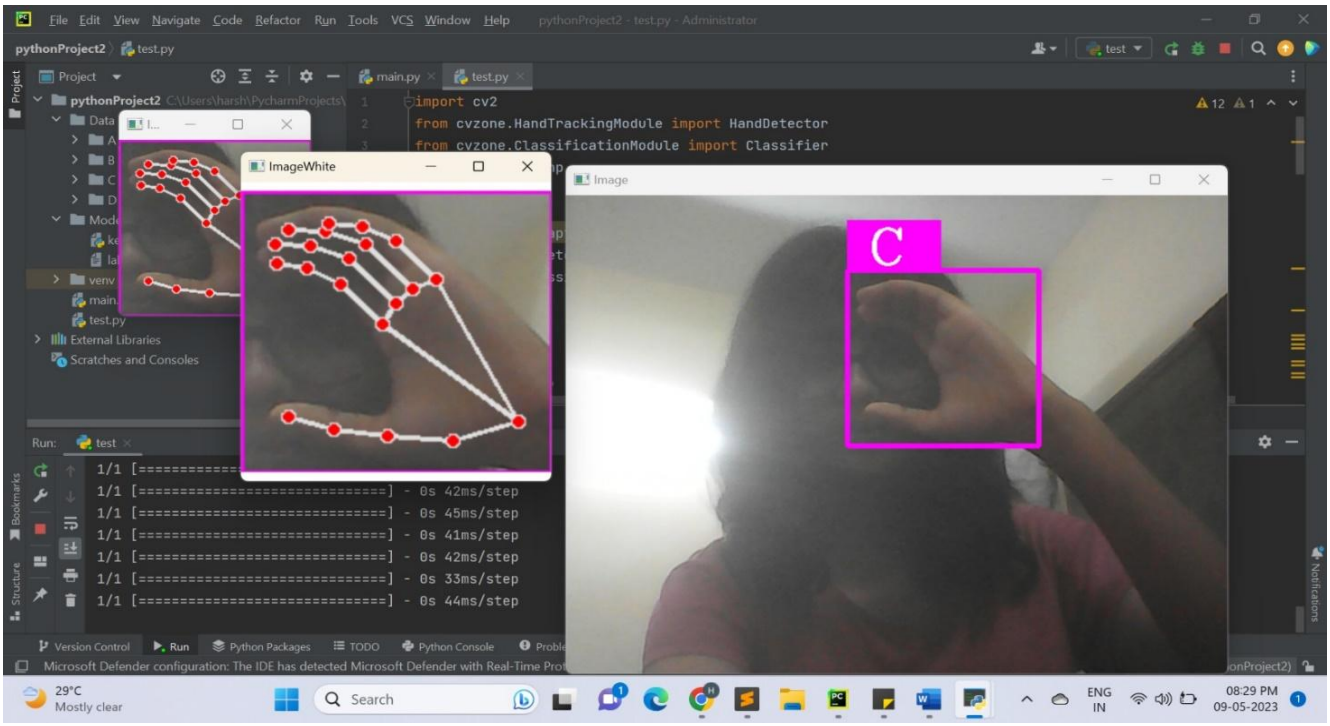
```
7172/7172 [==============================] - 0s 68us/step
Accuracy:  0.9408812046848857
```

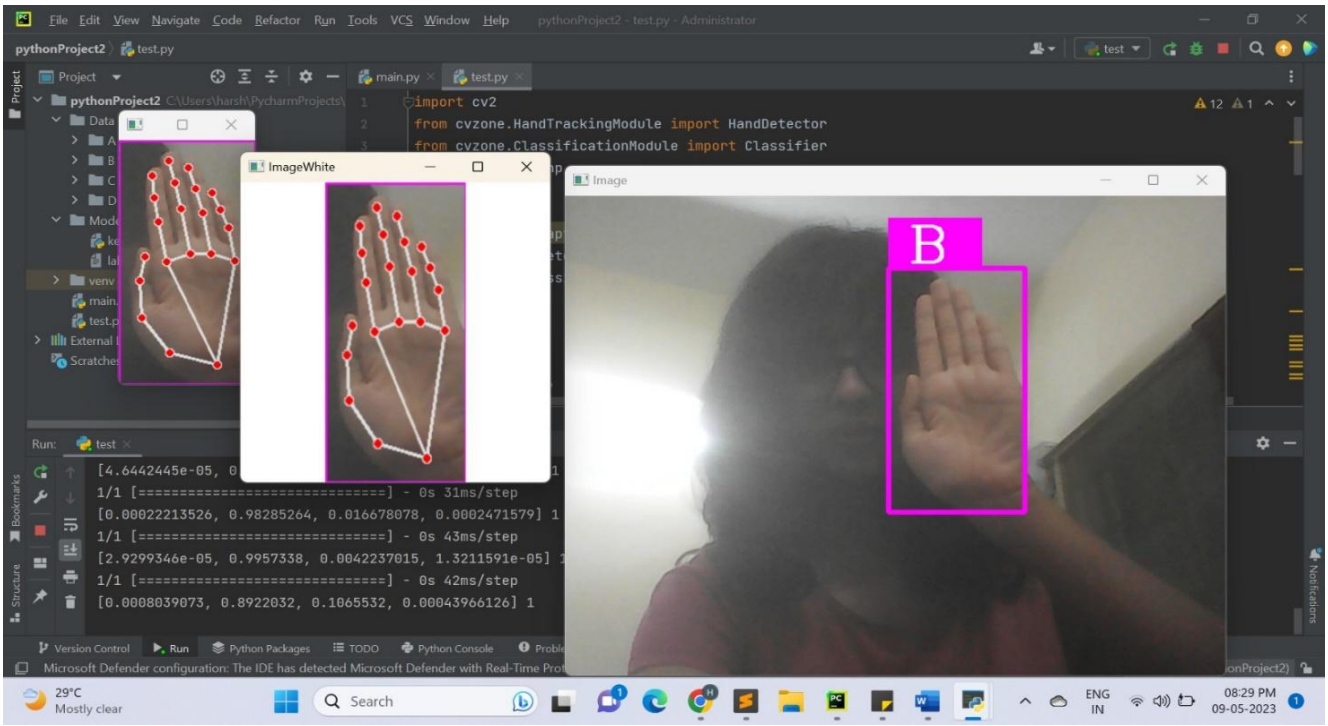
```
[17] classifier.summary()
```

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_1 (Conv2D)            (None, 28, 28, 8)         80
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 14, 14, 8)         0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 14, 14, 16)        1168
_________________________________________________________________
dropout_1 (Dropout)          (None, 14, 14, 16)        0
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 3, 3, 16)          0
_________________________________________________________________
dense_1 (Dense)              (None, 3, 3, 128)         2176
_________________________________________________________________
flatten_1 (Flatten)          (None, 1152)              0
_________________________________________________________________
dense_2 (Dense)              (None, 26)                29978
=================================================================
Total params: 33,402
Trainable params: 33,402
Non-trainable params: 0
_________________________________________________________________
```

```
[21] plot_model(classifier, show_shapes=True, show_layer_names=True, to_file='model.png')
     from IPython.display import Image
     Image(retina=True, filename='model.png')
```


140650033386104
conv2d_1: Conv2D
input: (None, 28, 28, 1)
output: (None, 28, 28, 8)
max_pooling2d_1: MaxPooling2D
input: (None, 28, 28, 8)
output: (None, 14, 14, 8)
conv2d_2: Conv2D
input: (None, 14, 14, 8)
output: (None, 14, 14, 16)
dropout_1: Dropout
input: (None, 14, 14, 16)
output: (None, 14, 14, 16)
max_pooling2d_2: MaxPooling2D
input: (None, 14, 14, 16)
output: (None, 3, 3, 16)
dense_1: Dense
input: (None, 3, 3, 16)
output: (None, 3, 3, 128)
flatten_1: Flatten
input: (None, 3, 3, 128)
output: (None, 1152)
dense_2: Dense
input: (None, 1152)
output: (None, 26)


ImageWhite
Image
C
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
1/1 [==========] - 0s 42ms/step
1/1 [==========] - 0s 45ms/step
1/1 [==========] - 0s 41ms/step
1/1 [==========] - 0s 42ms/step
1/1 [==========] - 0s 33ms/step
1/1 [==========] - 0s 44ms/step


ImageWhite
Image
B
[0.00022213526, 0.98285264, 0.016678078, 0.0002471579] 1
[0.0008039073, 0.8922032, 0.1065532, 0.00043966126] 1

# CONCLUSION AND FUTURE ENHANCEMENTS

## 7.1 conclusion

A sign language recognition model can be created using Python modules like OpenCV and Keras to help the disabled community. Here, a collection of inputs, such photos, are used to train the created model before it is used to generate a set of messages after recognising a particular hand motion.

This model has the unique ability to be trained to create a variety of alphabets using custom hand gestures, thereby bridging the communication gap between people who require special assistance and the general public.

## 7.2Future Enhancements

One major enhancement is the increased number of signs and messages. Since our model generates messages only for 4 different alphabets, we want our model to be able to generate a complete group of alphabets. Another enhancement is that we want to convert this idea to a web or an android application so that it can be used by a wide variety of people.

# REFERENCES

[1] Aya Hassonueh, and A.M.Mutawa, “Development of a Real-Time Emotion Recognition System Using Facial Expressions and EEG”, March 2020.

[2] José Herazo, “Sign language recognition using deep learning”, August 2020.

[3]K Abhijith Bhaskaran, Anoop G. Nair, K Deepak Ram, Krishnan Ananthanarayanan and H.R. Nandi Vardhan, “Smart gloves for hand gesture recognition: Sign language to speech conversion system”, IEEE Conference, 2016.

[4]Sanil Jain and K.V.Sameer Raja, “Indian Sign Language Character Recognition” , Indian Institute of Kanpur, April 2019.

[5] Helen Cooper, Brian Holt, Richard Bowden: “Visual Analysis of Humans”, 2011.

[6] Aditya Das, Shantanu Gawde, Khyati Suratwala and Dhananjay Kalbande, “Sign