# Real-Time Sign Language Recognition for Enhanced Communication


Team No : 1

Team Details:

1. Ch.Sweekruthi 21eg505810
2. L.Sreepadh 21eg505845
3. T.Meghanath 21eg505865

Project Supervisor
Mrs. B.Ujwala
Assistant Professor

# Introduction

- Deaf and Dumb people face a lot of problems in their daily lives and interactions. Communication is the only medium by which we can share our thoughts or convey the message but for a person with disability (deaf and dumb) faces difficulty in communication with normal person.

- Speech impaired people use hand signs and gestures to communicate. Normal people face difficulty in understanding their language. Hence there is a need of a system which recognizes the different signs, gestures and conveys the information to the normal people.

# Problem Statement

Speech impaired people use hand signs and gestures to communicate, Normal people face difficulty in understanding their language.

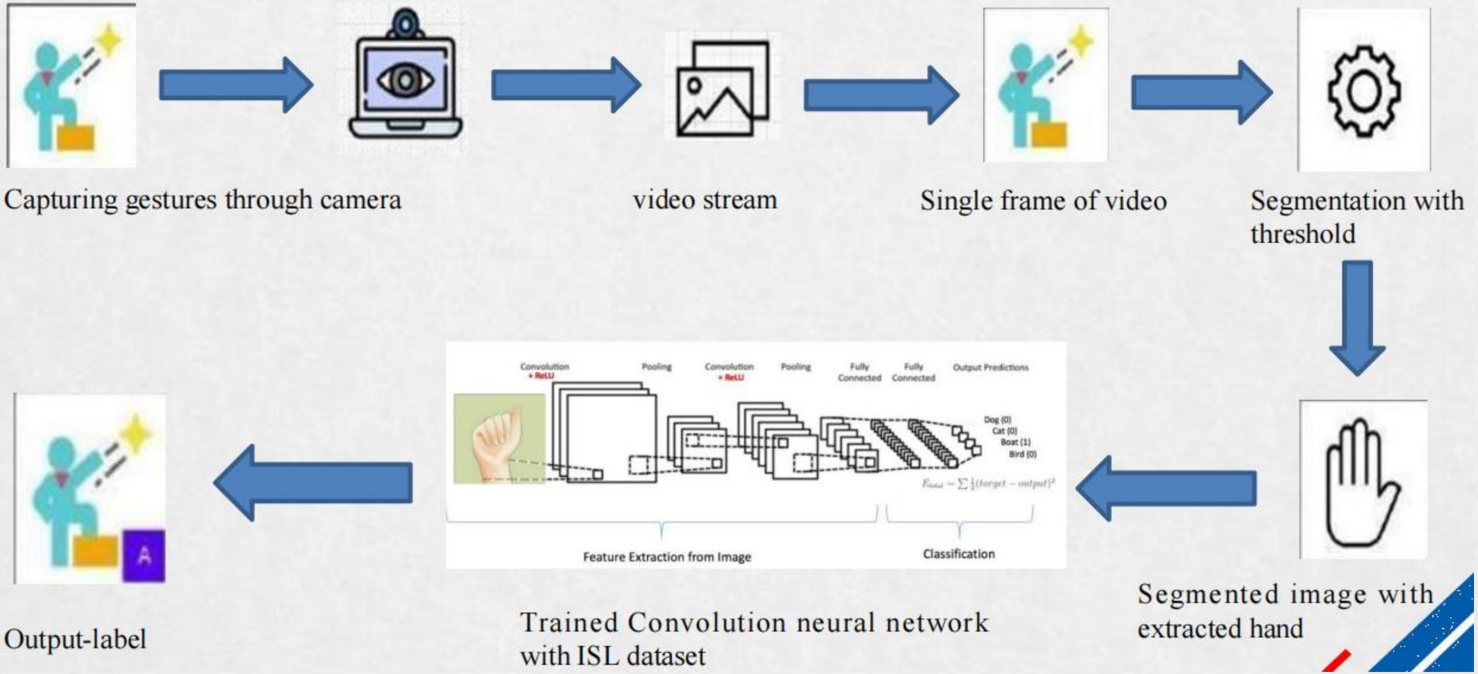
Building a robust real-time sign language recognition with high accuracy and accessibility, aimed at enhancing communication between Deaf and hard of hearing individuals and the wider society.

# Proposed Method

- Our proposed system is real-time sign language recognition system for enhanced communication, which uses the convolution neural network which recognizes various hand gestures by capturing video and converting it into frames.
- Then those hand pixels are segmented and the image which is obtained sent for comparision to the trained model.
- The hand gestures are of Indian Sign Language, the model is trained on the ISL.
- Thus our system is more robust in getting exact text labels of each letters

# Proposed Method

**System Architecture :**



# Experiment Environment

**Operating System :** Windows

**IDE :** Google Colab, VS Code

**Libraries:** Tensorflow, Keras, cv2, numpy, matplotlib

**Tensorflow & Keras**: TensorFlow and Keras, often used synonymously for deep learning tasks, are employed to build and train sign language recognition models. Model architectures are then designed using Keras, with TensorFlow as the backend, usually involving Convolutional Neural Networks (CNNs) for image recognition.

**cv2:** OpenCV (Open Source Computer Vision Library) is a popular library for computer vision and image processing tasks. It is used for capturing video frames, image manipulation, and drawing on images. In this code, it's used for capturing video from the camera, resizing images, and displaying images with annotations.

# Experiment Environment

**numpy:** NumPy is a fundamental library for numerical operations in Python. It's commonly used for array and matrix manipulation. In this code, it's used for various mathematical calculations and creating arrays.

**matplotlib:** Matplotlib is a popular library in Python for creating static, interactive visualizations and plots.

**Programming Languages:** Python 3.10

**Hardware Requirements:**
**Camera:** Good Quality, USB-2.0 HD
**Processor**: Intel i5
**RAM**: 8.00 GB
**System type**: 64-bit operating system

# Experiment Screenshots

signtrain.ipynb

File Edit View Insert Runtime Tools Help Last edited on September 19

+ Code + Text

```
import numpy as np
import pickle
import tensorflow as tf
import keras
from keras.models import Sequential
from keras.layers import Conv2D,MaxPooling2D,Dense,Flatten,Dropout
import matplotlib.pyplot as plt
from keras.layers import BatchNormalization
from keras.preprocessing import image
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.regularizers import l1, l2
```

```
from google.colab import drive
drive.mount('/content/gdrive')
```

```
Mounted at /content/gdrive
```

```
train_dir="/content/gdrive/MyDrive/sign/data"
generator = ImageDataGenerator()
train_ds = generator.flow_from_directory(train_dir,target_size=(224, 224),batch_size=32)
classes = list(train_ds.class_indices.keys())
```

```
Found 10840 images belonging to 36 classes.
```

```
from tensorflow.keras.preprocessing.image import ImageDataGenerator

image_gen = ImageDataGenerator(
    rescale=1./255,
    rotation_range=40,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest',
    validation_split=0.2  # 20% of the data will be used for validation
)

train_data_gen = image_gen.flow_from_directory(
+    batch_size=32,
    directory=train_dir,
    target_size=(224, 224),
    class_mode='categorical',
    subset='training'  # Specify 'training' to get the training subset
)

val_data_gen = image_gen.flow_from_directory(
    batch_size=32,
    directory=train_dir,
    target_size=(224, 224),
    class_mode='categorical',
    subset='validation'  # Specify 'validation' to get the validation subset
)
```

```
Found 8672 images belonging to 36 classes.
Found 2168 images belonging to 36 classes.
```

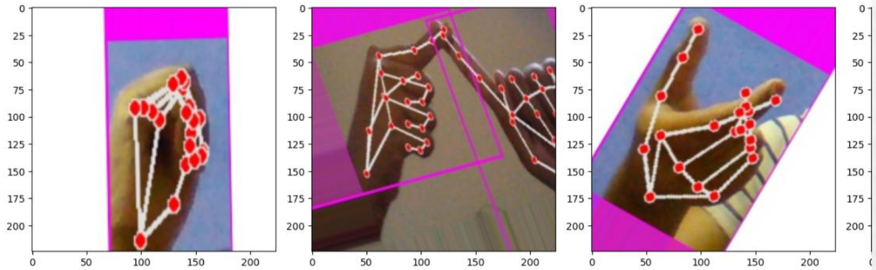
# Experiment Screenshots

signtrain.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```python
# This function will plot images in the form of a grid with 1 row and 5 columns where images are placed in each column.
def plotImages(images_arr):
    fig, axes = plt.subplots(1, 5, figsize=(20,20))
    axes = axes.flatten()
    for img, ax in zip(images_arr, axes):
        ax.imshow(img)
    plt.tight_layout()
    plt.show()

# Retrieve a batch of augmented images
augmented_images = [train_data_gen[0][0][i] for i in range(5)]

# Plot the augmented images using the plotImages function
plotImages(augmented_images)
```




signtrain.ipynb

File Edit View Insert Runtime Tools Help All changes saved

+ Code + Text

```python
model = Sequential()
model.add(Conv2D(32, kernel_size = (3, 3), activation='relu', input_shape=(224,224,3)))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(64, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(96, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Conv2D(32, kernel_size=(3,3), activation='relu'))
model.add(MaxPooling2D(pool_size=(2,2)))
model.add(BatchNormalization())
model.add(Dropout(0.2))
model.add(Flatten())
model.add(Dense(128, activation='relu'))
#model.add(Dropout(0.3))
model.add(Dense(len(classes),activation='softmax'))
```

```python
classes
```

```
['0',
 '1',
 '2',
 '3',
 '4',
 '5',
 '6',
 '7',
 '8',
```

# Experiment Screenshots

ANURAG UNIVERSITY

```
model.compile(loss = 'categorical_crossentropy',optimizer = 'adam',metrics = ["accuracy"])
model.summary()
```

```
Model: "sequential_3"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 conv2d_12 (Conv2D)          (None, 222, 222, 32)      896

 max_pooling2d_12 (MaxPooli  (None, 111, 111, 32)      0
 ng2D)

 batch_normalization (Batch  (None, 111, 111, 32)      128
 Normalization)

 conv2d_13 (Conv2D)          (None, 109, 109, 64)      18496

 max_pooling2d_13 (MaxPooli  (None, 54, 54, 64)        0
 ng2D)

 batch_normalization_1 (Bat  (None, 54, 54, 64)        256
 chNormalization)

 conv2d_14 (Conv2D)          (None, 52, 52, 64)        36928

 max_pooling2d_14 (MaxPooli  (None, 26, 26, 64)        0
 ng2D)

 batch_normalization_2 (Bat  (None, 26, 26, 64)        256
 chNormalization)

 conv2d_15 (Conv2D)          (None, 24, 24, 96)        55392

 max_pooling2d_15 (MaxPooli  (None, 12, 12, 96)        0
 ng2D)

 batch_normalization_3 (Bat  (None, 12, 12, 96)        384
```

```
epochs=5
history = model.fit(train_data_gen,steps_per_epoch=int(np.ceil(8672/ float(32))),epochs=epochs,validation_data=val_data_gen,validation_steps=int(np.ceil(2168/ float(32))))
```

```
Epoch 1/5
271/271 [==============================] - 1220s 4s/step - loss: 1.1899 - accuracy: 0.6233 - val_loss: 9.0324 - val_accuracy: 0.0360
Epoch 2/5
271/271 [==============================] - 959s 4s/step - loss: 0.4038 - accuracy: 0.8577 - val_loss: 0.7252 - val_accuracy: 0.7652
Epoch 3/5
271/271 [==============================] - 949s 4s/step - loss: 0.2384 - accuracy: 0.9192 - val_loss: 0.5325 - val_accuracy: 0.8413
Epoch 4/5
271/271 [==============================] - 947s 3s/step - loss: 0.1745 - accuracy: 0.9415 - val_loss: 0.4684 - val_accuracy: 0.8722
Epoch 5/5
271/271 [==============================] - 941s 3s/step - loss: 0.1368 - accuracy: 0.9516 - val_loss: 0.1885 - val_accuracy: 0.9308
```

```
final_train_accuracy = history.history['accuracy'][-1]
print(f'Final training accuracy: {final_train_accuracy * 100:.2f}%')
```

```
Final training accuracy: 98.78%
```

```
model.save('keras_model.h5')
```

# Experiment Screenshots

ANURAG UNIVERSITY

project > tes2.py > ...

```python
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math

cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=2)  # Detect up to 2 hands
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

offset = 20
imgSize = 300

labels = ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9", "A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "

while True:
    success, img = cap.read()
    imgOutput = img.copy()
    hands, img = detector.findHands(img)

    # Detect single-hand gestures
    if len(hands) == 1:
        hand = hands[0]
        x, y, w, h = hand['bbox']
        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]

        imgCropShape = imgCrop.shape
        aspectRatio = h / w

        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
```

# Experiment Screenshots

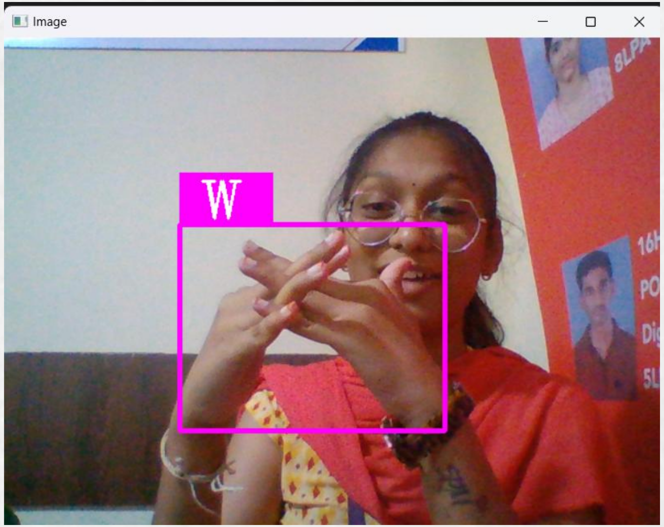
project > tes2.py > ...

```
            label = labels[index]
            confidence = prediction[index]
            print("Single-hand gesture:", label)

        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label = labels[index]
            confidence = prediction[index]
            print("Single-hand gesture:", label)

        cv2.rectangle(imgOutput, (x - offset, y - offset - 50),
                      (x - offset + 90, y - offset - 50 + 50), (255, 0, 255), cv2.FILLED)
        cv2.putText(imgOutput, label, (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (255, 0, 255), 4)

    # Detect double-hand gestures
    elif len(hands) == 2:
        hand1 = hands[0]
        hand2 = hands[1]

        # Calculate the combined bounding box that includes both hands
        x1, y1, w1, h1 = hand1['bbox']
        x2, y2, w2, h2 = hand2['bbox']
        x = min(x1, x2)
        y = min(y1, y2)
        w = max(x1 + w1, x2 + w2) - x
        h = max(y1 + h1, y2 + h2) - y

        imgWhite = np.ones((imgSize, imgSize, 3), np.uint8) * 255
        imgCrop = img[y - offset:y + h + offset, x - offset:x + w + offset]
```

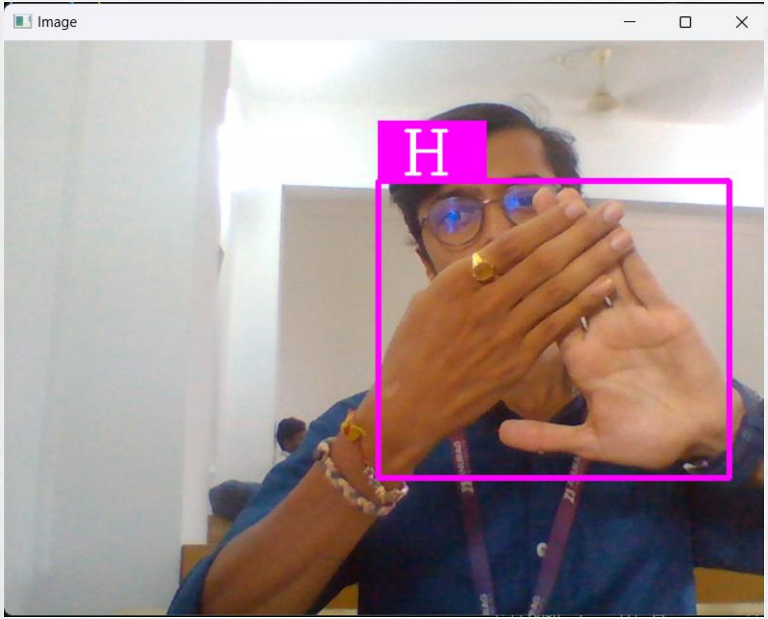
project > tes2.py > ...

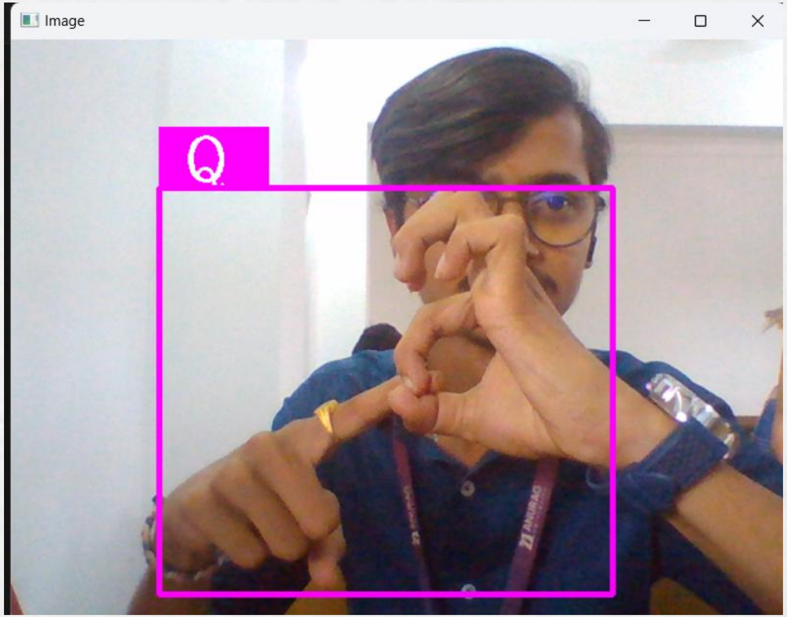
```

        imgCropShape = imgCrop.shape
        aspectRatio = h / w

        if aspectRatio > 1:
            k = imgSize / h
            wCal = math.ceil(k * w)
            imgResize = cv2.resize(imgCrop, (wCal, imgSize))
            imgResizeShape = imgResize.shape
            wGap = math.ceil((imgSize - wCal) / 2)
            imgWhite[:, wGap:wCal + wGap] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label = labels[index]
            confidence = prediction[index]
            print("Double-hand gesture:", label)

        else:
            k = imgSize / w
            hCal = math.ceil(k * h)
            imgResize = cv2.resize(imgCrop, (imgSize, hCal))
            imgResizeShape = imgResize.shape
            hGap = math.ceil((imgSize - hCal) / 2)
            imgWhite[hGap:hCal + hGap, :] = imgResize
            prediction, index = classifier.getPrediction(imgWhite, draw=False)
            label = labels[index]
            confidence = prediction[index]
            print("Double-hand gesture:", label)

        cv2.rectangle(imgOutput, (x - offset, y - offset - 50),
                      (x - offset + 90, y - offset - 50 + 50), (255, 0, 255), cv2.FILLED)
        cv2.putText(imgOutput, label, (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
        cv2.rectangle(imgOutput, (x - offset, y - offset), (x + w + offset, y + h + offset), (255, 0, 255), 4)

    cv2.imshow("Image", imgOutput)
    cv2.waitKey(1)

```

# Experiment Results

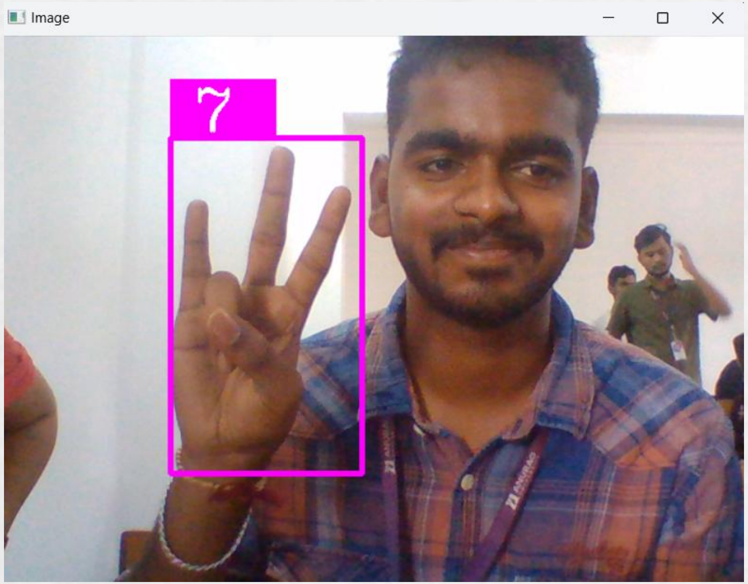



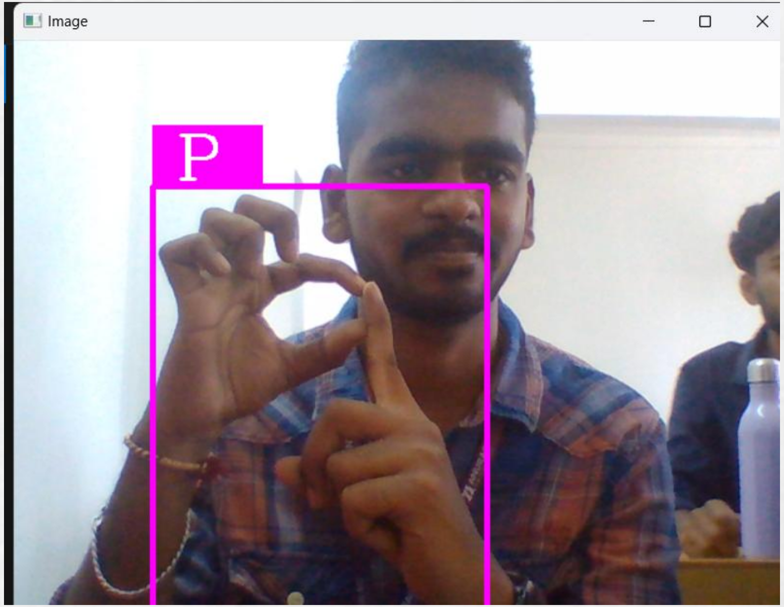
# Experiment Results



# Experiment Results



# Finding

```
Epoch 1/5
271/271 [==============================] - 1220s 4s/step - loss: 1.1899 - accuracy: 0.6233 - val_loss: 9.0324 - val_accuracy: 0.0360
Epoch 2/5
271/271 [==============================] - 959s 4s/step - loss: 0.4038 - accuracy: 0.8577 - val_loss: 0.7252 - val_accuracy: 0.7652
Epoch 3/5
271/271 [==============================] - 949s 4s/step - loss: 0.2384 - accuracy: 0.9192 - val_loss: 0.5325 - val_accuracy: 0.8413
Epoch 4/5
271/271 [==============================] - 947s 3s/step - loss: 0.1745 - accuracy: 0.9415 - val_loss: 0.4684 - val_accuracy: 0.8722
Epoch 5/5
271/271 [==============================] - 941s 3s/step - loss: 0.1368 - accuracy: 0.9516 - val_loss: 0.1885 - val_accuracy: 0.9308
```

```
final_train_accuracy = history.history['accuracy'][-1]
print(f'Final training accuracy: {final_train_accuracy * 100:.2f}%')
```

```
Final training accuracy: 98.78%
```

- While detecting the Sign Language, it is noted that the model convolutional neural networks have the accuracy of 98% in which it is much greater than the machine learning models such as Decision Tree, Naive Bayes, KNN Classification.

# Justification

1.What are parameters improved by your method?

- Accuracy

2. Why your parameter values improved?

- Quality of Dataset
- Data Augumentation
- Number of Epochs