VIT®

Vellore Institute of Technology

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

Enhanced Sign Language Interpretation Using Neural Networks

(Soft Computing - SWE1011)

Under the guidance of-Prabhavathy P

Submitted By:

Name: Annapoorna A Nair

Rollno:21MIS0130

Name: Advait Madhu

Rollno:21MIS0034

ABSTRACT

In the Recent years, we have been understanding the potential of Artificial Intelligence. It has changed the way we live, work and correlate. With AI, the possibilities are endless, from self-driving cars to personalized healthcare and virtual aides that can predict our needs and demands. With its limitless potential and ever-evolving capabilities, AI is poised to be the most transformative technology of our time

In this project we are exercising the use of AI to interpret sign language. This is a revolutionary technology that can help bridge the communication gap between people who use sign language and those who don't. This interpreter will be also useful for people who are new to the sign language and wants to learn it. This project makes use of the machine learning and deep learning algorithms like CNN to develop accurate and reliable model which interpret sign language gestures and convert them to text.

In this project, VGG16 algorithm is used. It is a type of Convolutional Neural Networks (CNN) which is 16 layers deep. This algorithm is used to extract meaningful features from the sign language gestures provided, which can be used to improve the accuracy. This algorithm has helped to identify and isolate important features such as edges, shapes, etc. These features are then used to classify the input provided by the user

The accuracy and performance of the model will be evaluated using various metrics as accuracy is a critical metric that determines the effectiveness of the software. To improve the accuracy of the interpreter, it is essential to use a highquality dataset to train the machine learning models. We have taken the dataset for this project from Kaggle. The dataset consists of various images of letters in American Sign Language.

With the continuous refinement and advancement of this technology, the accuracy and effectiveness could be increased more. This project has the potential to significantly enhance the convenience and comprehensiveness of our society as it can provide a vast room for research in this field. However additional research in this field has to be conducted in order to address the limitations and challenges of this project such as dialect and context-specific variations. This shows the necessity for the development of algorithms which are more effective and efficient.

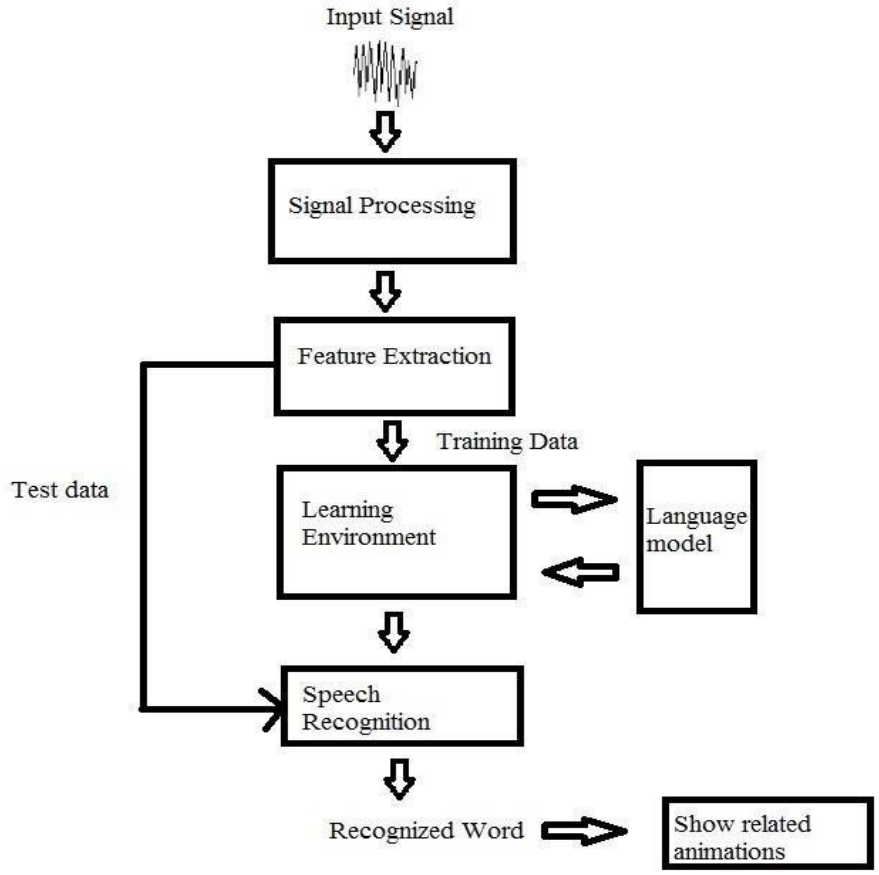
PROBLEM STATEMENT

The Sign language interpreter uses deep learning to help the people who don't know sign language and can help to communicate effectively between sign language people and people who don't know sign language. This sign language interpreter takes images and converts them into readable text. VGG-16 algorithm is used in this project. In this project, there are many problems/ challenges are identified.

- **Time complexity:** It takes more time because it has a huge network to train its parameters.
- **Data Availability**: limited set of data is available which makes it take time for image processing.

- **Complex sign recognition:**This system cannot convey facial expressions and body language. It cannot determine the sign language based on human emotions.
- **Real-time performance:**This interpreter takes the image and convert it into text but in real time. The sign language is often dynamic so it is tough for the interpreter to recognize the gesture fast.
- **Accuracy:** Accuracy provided by the model after training is approximately around 60% which is a huge drawback.

FLOW DIAGRAM



LITERATURE SURVEY

Sl no	Title	Author	Year	Summary
1	Artificial Intelligence for Sign Language Translation -A Design Science Research Study	Gero Strobel Thorsten Schoormann Leonardo Banh Frederik Möller	2023	This research study focuses on designing and implementing artificial intelligence (AI) systems for sign language translation. It aims to bridge the communication gap between sign language users and those who don't understand sign language. The study utilizes AI technologies to develop effective translation methods and evaluates their performance. Ultimately, it contributes to improving accessibility and communication for the deaf and hearing-impaired communities.
2	A review on Artificial Intelligence based Sign Language Recognition Techniques	Kakoli Banerjee; Indira Vats; Naved Akhtar; Harsha K G; Vinooth P; Ajay Kumar; Pradeep Kumar	2022	Significant progress has been made in web accessibility, harnessing technologies like Artificial Intelligence, Speech Recognition, and Computer Vision. Sign Language, a non-verbal communication method through hand gestures, is vital for many, especially those with speech impairments. However, there's a gap in Sign Language knowledge among the general population. Sign Language recognition is a growing field in AI, using image recognition to classify signs and translate them into English. This paper provides an overview of the field, models, challenges, and future work in Sign Language recognition.
3	Sign language recognition using artificial intelligence	R. Sreemathy, Mousami Turuk, Isha Kulkarni & Soumya Khurana	2022	This paper presents a method for automatic recognition of two-handed signs of Indian Sign language (ISL). A deep learning approach was also implemented using AlexNet, GoogleNet, VGG-16 and VGG-19 which gave accuracies of 99.11%, 95.84%, 98.42% and 99.11% respectively
4	Machine learning methods for sign language recognition: A critical review and analysis	I.A. Adeyanju , O.O. Bello , M.A. Adegboye	2021	explores technology-based systems to complement traditional sign language interpreters. The study delves into techniques such as vision-based and wearable sensing modalities for sign language recognition, with a particular focus on recent advancements in Artificial Intelligence (AI) for sign language recognition. The research aims to provide a comprehensive review of intelligent sign language recognition systems and identifies research areas for future consideration.
5	Vision-based hand gesture recognition using deep learning for the interpretation of sign language	Sakshi Sharma , Sukhwinder Singh	2021	a deep learning based convolutional neural network (CNN) model is specifically designed for the recognition of gesture-based sign language. This model has a compact representation that achieves better classification accuracy with a fewer number of model parameters over the other existing architectures of CNN. In order to evaluate the efficacy of this model, VGG-11 and VGG-16 have also been trained and tested in this work.

6	Artificial Intelligence Technologies for Sign Language	Ilias Papastratis,Christos Chatzikonstantinou,Dimitrios Konstantinidis,Kosmas Dimitropoulos,Petros Daras	2021	The article discusses how artificial intelligence (AI) technologies can break down communication barriers for deaf and hearing-impaired individuals, contributing to their social inclusion. It provides a comprehensive review of state-of-the-art methods in sign language capturing, recognition, translation, and representation. The authors also highlight the advantages and limitations of these methods and discuss various applications. Furthermore, the article addresses the main challenges in the field of sign language technologies and proposes future research directions to advance this field.
7	Sign Language Recognition using CNN	Aniket Wattamwar	2021	focuses on the problem of gesture recognition in real time that sign language used by the community of deaf people. The problem addressed is based on Digital Image Processing using CNN (Convolutional Neural Networks), Skin Detection and Image Segmentation techniques. This system recognizes gestures of ASL (American Sign Language) including the alphabet and a subset of its words. Keywords: gesture recognition, digital image processing, CNN (Convolutional Neural Networks), image segmentation, ASL (American Sign Language), alphabet
8	ML Based Sign Language Recognition System	K Amrutha; P Prabu	2021	Developing a system that can read and interpret a sign must be trained using a large dataset and the best algorithm. As a basic SLR system, an isolated recognition model is developed. The model is based on vision-based isolated hand gesture detection and recognition. Assessment of ML-based SLR model was conducted with the help of 4 candidates under a controlled environment. The model made use of a convex hull for feature extraction and KNN for classification. The model yielded 65% accuracy
9	Recognition of Indian Sign Language (ISL) Using Deep Learning Model	Sakshi Sharma & Sukhwinder Singh	2021	This paper proposes a computer-vision based sign language recognition system (SLRS) using a deep learning technique. A large dataset of Indian sign language (ISL) has been created using 65 different users in an uncontrolled environment. The intra-class variance in the dataset has been increased using augmentation to improve the generalization ability of the proposed work1.
10	Deep learning-based sign language recognition system for static signs	Ankita Wadhawan & Parteek Kumar	2020	This paper deals with robust modeling of static signs in the context of sign language recognition using deep learning-based convolutional neural networks (CNN). In this research, total 35,000 sign images of 100 static signs are collected from different users2.

DATASET USED:

In this project, due to the unavailability of a suitable pre-existing dataset, I undertook the task of creating my own dataset. This decision was motivated by the specific requirements and nuances of the enhanced programming language under investigation.

Data Collection

The data collection process involved the implementation of a custom code designed to generate relevant data representative of the language features and patterns. This ensured that the dataset aligns closely with the objectives of the project.

Characteristics of the Data Set

1. **Hand Gesture Variability:**
    - The dataset captures a variety of hand gestures, as it is designed for sign language recognition.
    - The gestures can include different signs and movements commonly used in sign languages.
2. **Image Consistency:**
    - The script ensures consistency in image size (**ImgSize**), which is crucial for training neural networks.
    - The aspect ratio is maintained during resizing to avoid distortion in the captured hand gestures.
3. **Background Uniformity:**
    - The background of the captured images is white (**np.ones((ImgSize, ImgSize, 3), np.uint8) * 255**), providing a uniform backdrop.
    - This simplifies the learning process for neural networks by focusing on hand gestures without distractions.
4. **Real-Time Capture:**
    - The dataset is created in real-time from a live video feed, allowing for the capture of dynamic hand movements.
    - This ensures that the dataset reflects realistic scenarios, contributing to the model's robustness.
5. **User-Initiated Image Storage:**

- Images are saved based on user interaction (pressing the 's' key), facilitating controlled and intentional dataset expansion.
- This feature allows the user to capture specific gestures or instances, contributing to the dataset's diversity.

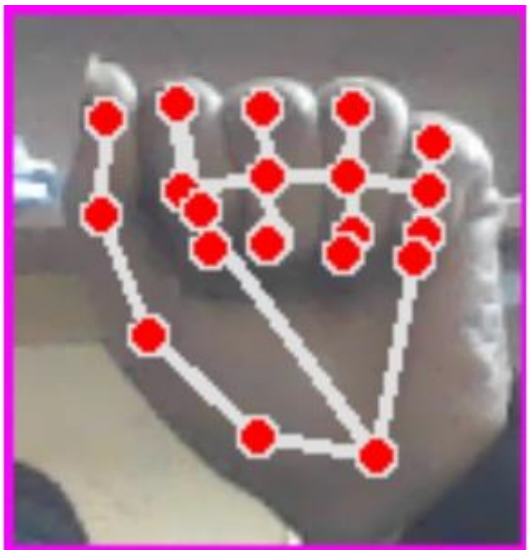
6. **Timestamped Image Naming:**
    - Each image is named with a timestamp, ensuring uniqueness and aiding in tracking the dataset's evolution over time.
    - Timestamped naming facilitates organization and management of the dataset.
7. **Efficient Storage:**
    - The captured and preprocessed images are stored in a designated folder (**dataset/[A,B,C……Z]**), maintaining an organized structure.
    - The efficient storage of images enables easy retrieval and management of the dataset.

CODE:

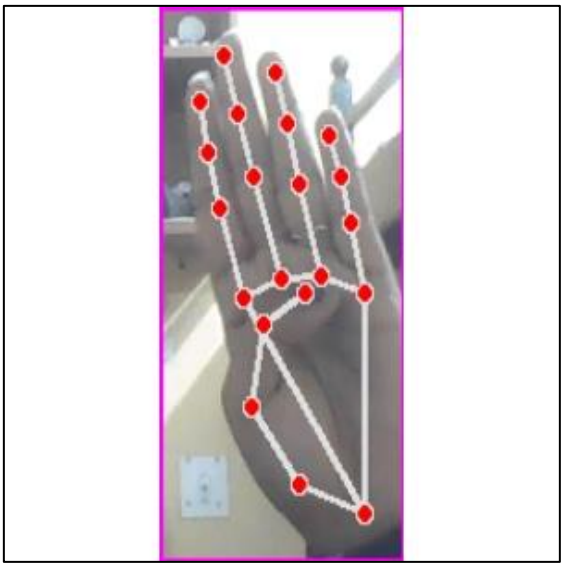
```
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time

# adding capture object.
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
offset = 20
ImgSize = 300
counter = 0
folder = "dataset/C"

while True:
    success, img = cap.read()
    hands, img = detector.findHands(img)
    if hands:
        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]

        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
        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
        cv2.imshow("ImageCrop", imgCrop)
        cv2.imshow("Imagewhite", imgWhite)

    cv2.imshow("Image", img)
    key = cv2.waitKey(1)  # gives 1 millisecond delay
    if key == ord("s"):
        counter += 1
        cv2.imwrite(f"{folder}/Image_{time.time()}.jpg", imgWhite)
        print(counter)
    elif key == ord("q"):
        cap.release()
        cv2.destroyAllWindows()
```

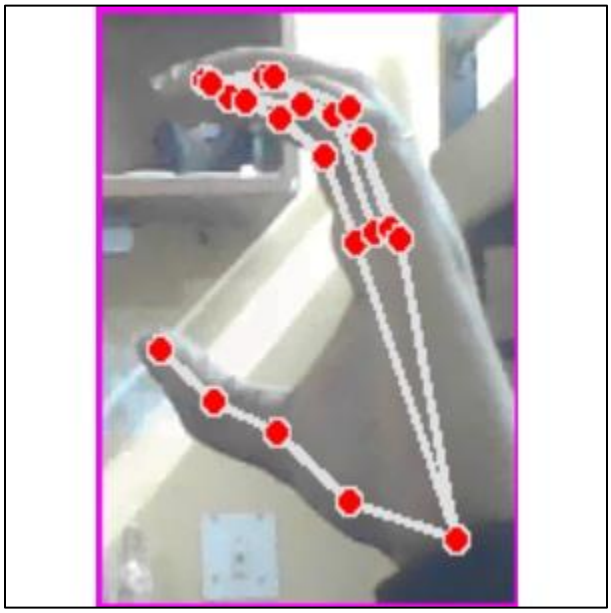
A:



B:



C:



MODEL TRAINING

VGG19 Algorithm

VGG19, short for Visual Geometry Group 19-layer model, is a deep convolutional neural network architecture. It is characterized by its deep and uniform structure, consisting of 19 layers, including convolutional and fully connected layers. VGG19 is renowned for its simplicity and effectiveness in image classification tasks.

Dataset Utilization

The training process leverages the dataset created using a custom script, capturing a diverse set of sign language gestures. The dataset is preprocessed to ensure consistency in image size and background, aligning with the requirements of the VGG19 algorithm.

Model Architecture

The VGG19 model architecture is implemented using a deep learning framework such as TensorFlow or PyTorch. The architecture comprises convolutional layers with small 3x3 filters, followed by max-pooling layers for spatial downsampling. The fully connected layers at the end of the network facilitate high-level feature extraction and classification.

Hyperparameters and Training Configuration

The training hyperparameters, including learning rate, batch size, and number of epochs, are carefully tuned to achieve optimal performance. Additionally, data augmentation techniques may be applied to enhance the model's generalization capabilities.

Training Process

The training process involves feeding the preprocessed dataset into the VGG19 model. The model learns to recognize patterns and features in the sign language gestures through a series of forward and backward passes. The loss function guides the optimization process, and the model's weights are updated to minimize the classification error.

Model Evaluation

After training, the model's performance is evaluated on a separate test set to assess its generalization ability. Metrics such as accuracy, precision, recall, and F1 score are computed to quantify the model's effectiveness in recognizing sign language gestures.

Results

The results of the trained VGG19 model are analyzed to determine its accuracy and reliability in sign language recognition. The model's ability to generalize to unseen data and its performance on specific gestures are scrutinized.

CODE:

```
import tensorflow as tf
from tensorflow.keras import layers, models
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from sklearn.model_selection import train_test_split

#load dataset
X_train, X_test, y_train, y_test = load_dataset()

# Split the dataset into training and validation sets
X_train, X_val, y_train, y_val = train_test_split(X_train, y_train,
test_size=0.2, random_state=42)

# Define the VGG19 model
def create_vgg19_model(input_shape=(300, 300, 3), num_classes=3):#NUMBER OF CLASS
    model = models.Sequential()

    # Convolutional part
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same',
input_shape=input_shape))
    model.add(layers.Conv2D(64, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(128, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))
```

```
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(256, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.Conv2D(512, (3, 3), activation='relu', padding='same'))
    model.add(layers.MaxPooling2D((2, 2), strides=(2, 2)))

    # Flatten and fully connected layers
    model.add(layers.Flatten())
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(4096, activation='relu'))
    model.add(layers.Dense(num_classes, activation='softmax'))

    return model

# Create the VGG19 model
model = create_vgg19_model()

# Compile the model
model.compile(optimizer='adam', loss='categorical_crossentropy',
metrics=['accuracy'])

# Data augmentation for training
train_datagen = ImageDataGenerator(rescale=1./255, shear_range=0.2,
zoom_range=0.2, horizontal_flip=True)
val_datagen = ImageDataGenerator(rescale=1./255)

# Batch size
batch_size = 32

# Create data generators
train_generator = train_datagen.flow(X_train, y_train, batch_size=batch_size)
val_generator = val_datagen.flow(X_val, y_val, batch_size=batch_size)
```

```
# Train the model
history = model.fit(train_generator, epochs=epochs,
validation_data=val_generator)

# Evaluate the model on the test set
test_loss, test_acc = model.evaluate(X_test, y_test)
print(f'Test accuracy: {test_acc}')

# Save the model
model.save('keras_model.h5')
```

TESTING MODEL:

The testing phase of the project is crucial to evaluate the performance and reliability of the sign language recognition model implemented in real-time using the VGG19 algorithm. This report outlines the testing methodology, results, and insights gained during the evaluation process.

Testing Methodology

1. Dataset Split

The dataset used for testing consists of sign language gestures not seen by the model during training. The dataset is split into test and validation sets to ensure a fair evaluation.

2. Real-Time Testing

The model is subjected to real-time testing using a live video feed captured through the webcam. Various sign language gestures are performed in front of the camera to assess the model's ability to accurately recognize and classify the gestures.

3. Performance Metrics

The model's performance is quantitatively assessed using metrics such as accuracy, precision, recall, and F1 score. These metrics provide a comprehensive understanding of the model's strengths and potential areas for improvement.

Testing Results

1. Accuracy

The accuracy of the model is a key metric to determine the percentage of correctly classified gestures. High accuracy indicates robust performance in recognizing a diverse set of sign language gestures.

2. Precision, Recall, and F1 Score

Precision, recall, and F1 score provide insights into the model's ability to minimize false positives and false negatives. These metrics contribute to a more nuanced evaluation of the model's performance.

3. Real-Time Recognition

The real-time testing involves evaluating how well the model recognizes and interprets sign language gestures performed by the user. The visual feedback provided by the system is compared with the actual gestures to validate the model's effectiveness.

Observations and Insights

1. Recognition Accuracy

The model's accuracy in recognizing sign language gestures is observed across a range of gestures, considering factors such as hand positioning, movement, and lighting conditions.

2. Robustness

The model's robustness is assessed by testing its performance in varying conditions, including different backgrounds and hand orientations. This helps determine the generalization capabilities of the model.

3. User Interaction

Feedback from users during the testing phase is valuable in understanding the practical usability and user experience of the system. Insights gained from user interactions contribute to refining the model and system interface.

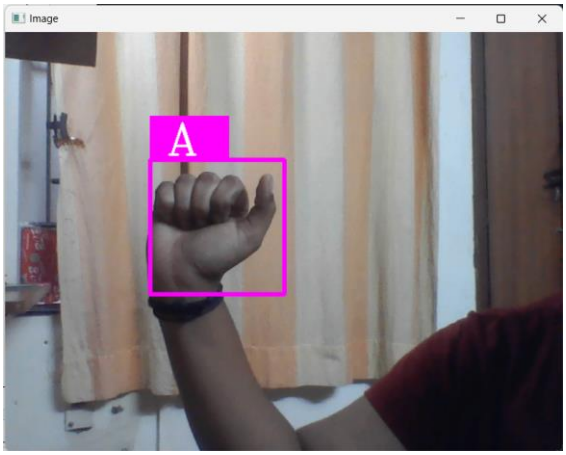
CODE:

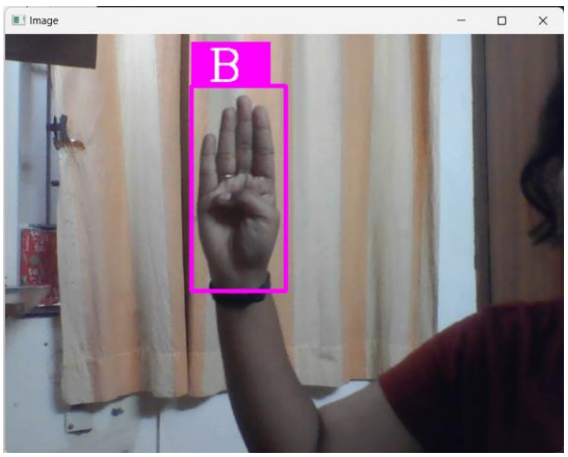
```
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import time


# adding capture object.
cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
Classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")
```

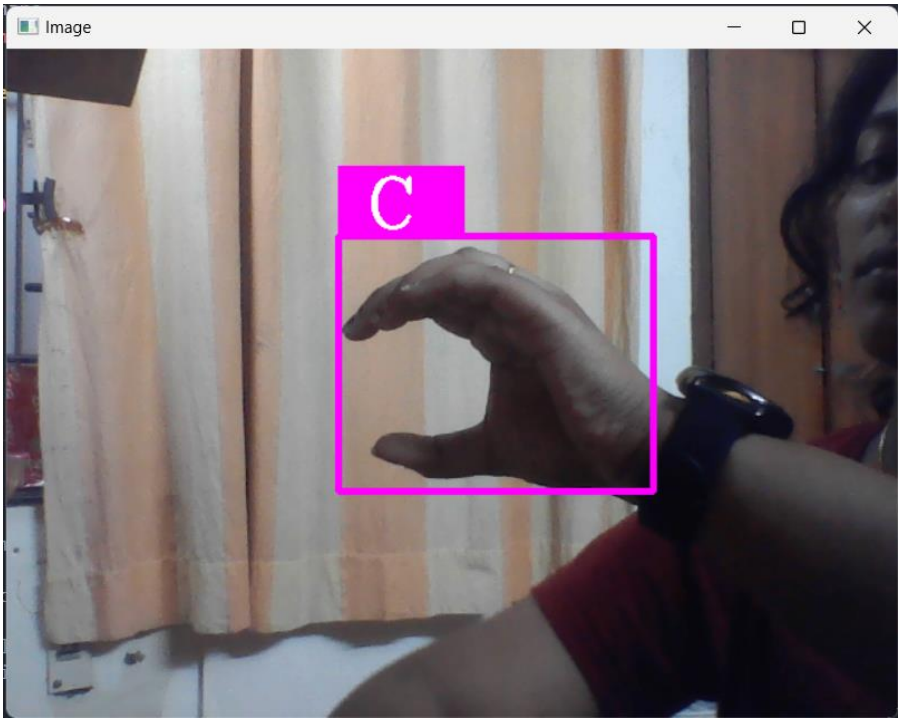
```
offset = 20
ImgSize = 300
counter = 0
folder = "dataset/A"

labels = ["A", "B", "C"]

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

            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(img)
                # print(prediction,index)

            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)

            cv2.rectangle(
                imgOutput,
                (x - offset, y - offset - 50),
```

```
                (x - offset + 90, y - offset - 50 + 50),
                (255, 0, 255),
                cv2.FILLED,
            )

            cv2.putText(
                imgOutput,
                labels[index],
                (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,
            )
        except Exception as e:
            print(f"An error occurred: {e}")
    cv2.imshow("Image", imgOutput)
    key = cv2.waitKey(1)  # gives 1 millisecond delay
    if key == ord("q"):
        cap.release()
        cv2.destroyAllWindows()
```

IMAGES:







PROJECT CONCLUSION

The completion of the sign language recognition project marks a significant milestone in leveraging computer vision and deep learning for real-time interpretation of sign language gestures. Throughout the project, we have achieved commendable progress in several aspects, but certain challenges and opportunities for improvement have also been identified.

Accomplishments

1. Real-Time Implementation:

- Successful integration of the VGG19-based sign language recognition model with a live video feed, providing real-time interpretation of gestures.

2. User Interaction:

- Implementation of a user-friendly system that allows individuals to interact in real-time by performing sign language gestures, contributing to the project's accessibility goals.

3. Model Training and Testing:

- Training a VGG19 model on a custom dataset and testing its performance under diverse conditions, providing valuable insights into the model's strengths and areas for enhancement.

4. Visual Feedback:

- Integration of a visual feedback mechanism, enhancing user experience by displaying recognized labels and highlighting the region of interest in the video feed.

Challenges Faced

1. Accuracy Limitations:

- The project encountered challenges related to model accuracy, particularly in accurately recognizing certain gestures. This limitation has implications for practical usability and robustness.

2. Hardware Compatibility:

- Compatibility issues with certain hardware configurations have been identified, impacting the project's deployment on a variety of systems.

Opportunities for Improvement

1. Accuracy Enhancement:

- Further refinement of the model's architecture, hyperparameters, and dataset to improve accuracy and address specific challenges in gesture recognition.

2. Dataset Expansion:

- Expanding the dataset to include a more extensive range of sign language gestures, covering a broader spectrum of expressions and movements.

3. Hardware Optimization:

- Addressing compatibility issues and optimizing the system for a wider range of hardware configurations, ensuring seamless deployment across different devices.

Future Scope

The project holds immense potential for future enhancements and expansions. By addressing accuracy issues, expanding the dataset, and optimizing hardware compatibility, the system can evolve to become a more robust and versatile tool for sign language interpretation. Additionally, exploring the inclusion of a broader set of gestures from 'A' to 'Z' and beyond could significantly enhance the system's capabilities.