SMART INTERNZ – PROJECT REPORT

AI-Powered Real-Time Communication System for People with Disabilities

SUBMITTED BY

A.YASEEN	(20481A1201)
G.DHANRAJ	(20481A1246)
G.DINESH KUMAR	(20481A1254)
G.KIRANN MAHY	(20485A1257)

SESHADRI RAO GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Table of Contents

1. INTRODUCTION

In a world that increasingly relies on seamless communication and connectivity, individuals with disabilities often face significant barriers in accessing and participating in real-time communication platforms. To address this pressing issue and promote inclusivity, we undertook the development of a groundbreaking project titled "Real-Time Communication System for People with Disabilities."

The primary objective of this project was to create an innovative communication platform tailored specifically to cater to the unique needs and challenges faced by people with disabilities. By leveraging cutting-edge technologies and incorporating user-centered design principles, our real-time communication system aimed to empower individuals with disabilities, enabling them to interact, share information, and engage in social activities with unprecedented ease and efficiency.

Moreover, we present the results of extensive user testing, involving individuals with disabilities who actively engaged with our machine learning-based communication system. Their valuable feedback and insights provide crucial validation of the system's efficacy, usability, and potential to enhance the quality of life for our target audience.

1.1 OVERVIEW

Our project aims to develop a real-time communication system using Artificial Intelligence (AI) to empower individuals who are deaf and mute. The system will leverage AI algorithms to interpret sign language and convert it into text or speech output, facilitating seamless communication for those with speech and hearing impairments. Through user-centered design and collaboration with the deaf and mute community, we seek to create an inclusive and transformative platform that enhances their lives and fosters greater connectivity with the world. This project represents a beacon of hope, promising innovative solutions to bridge communication gaps and make everyday tasks more accessible for individuals of all abilities.

1.2 PURPOSE

Our project seeks to empower individuals who are deaf and mute by harnessing the potential of artificial intelligence (AI) to enable effective communication. We recognize the power of speech in connecting people and sharing ideas, but we also acknowledge the unfairness faced by those who lack this gift. Through AI-driven technology, we aim to bridge communication gaps and ensure that individuals with disabilities are included in society.

By using convolutional neural networks and deep learning, we are developing a model that can understand various hand motions, enabling communication through sign language. This model will be integrated into an app, allowing deaf and mute individuals to express themselves and communicate with others effectively.

Our purpose is to create a more inclusive society, where technology facilitates accessibility for all, regardless of their abilities. Through this project, we aspire to make the world a place where everyone's voice is heard, and individuals with disabilities can engage actively in daily interactions and experiences.

2. LITERACY SURVEY

2.1 EXISTING PROBLEM

The issue at hand revolves around the need to improve communication for deaf and mute individuals by leveraging sign language recognition technology. Recent studies have demonstrated the potential of this technology to bridge the communication gap between specially abled individuals and the wider community.

Researchers have explored various approaches to recognize hand gestures represented through video feeds or images. For instance, Taniya Sahana et.al proposed a system based on multiscale density features, achieving 98.2% accuracy using the MLP model. Another study focused on Indian hand gestures, utilizing blob detection, skin color recognition, and template matching to classify 26 classes of gestures with a mobile application.

One notable technique involves using deep convolutional neural networks, such as AlexNet and VGGNet, to process hand images after tracking and recognition. Such efforts aim to enable two-sided communication efficiently for deaf and blind individuals.

Several solutions are currently in use, including technology like smartphones and laptops for text-based communication, sign language interpreters for fluent communication in sign language, and deaf individuals with residual hearing communicating with blind individuals using speech.

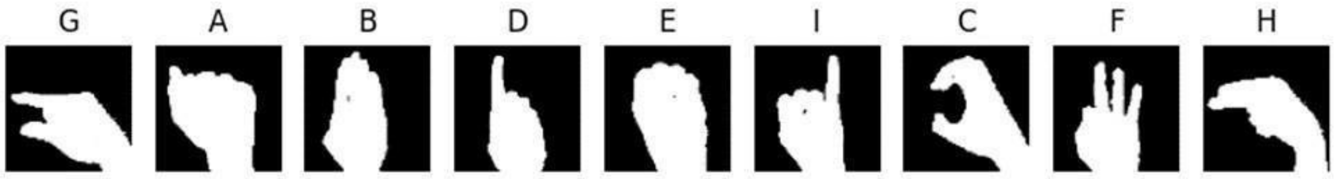
2.2 PROPOSED SOLUTION

Our project aims to create an innovative solution to bridge the communication gap between the speech and hearing impaired individuals and the wider community. The key objectives of our research are as follows:

2. **2.1Communication Gap Reduction:** We strive to design and develop a system that facilitates seamless communication between speech and hearing impaired individuals and the rest of the world. By leveraging advanced computer vision and deep learning techniques, we aim to enable effective interaction and understanding.

2.2.2 Deaf-Dumb Communication: Our proposed communication system will act as a bridge between deaf and mute individuals and those who can communicate verbally. Through the use of hand gestures, captured and classified using a convolutional neural network (CNN), our system will facilitate effective communication and understanding.

To address overfitting and enhance model performance, we freeze some initial layers of the VGG16 base model, reducing its complexity. Additional layers of dropout and dense are added to the output layer after flattening, allowing the model to predict the correct class among the nine hand gestures representing letters from 'A' to 'I'.



3. THEORITICAL ANALYSIS

3.1 BLOCK DIAGRAM

3.1.1Image Dataset:

This is the collection of images that you will use to train and test our neural network model. The dataset contains hand gesture images representing letters from 'A' to 'I' used for classification.

3.1.2. Image Preprocessing:

Before feeding the images into the neural network, you need to preprocess them to ensure they are in a suitable format for training. Preprocessing may include resizing, normalization, and other image transformations.

3.1.3. Train Data and Test Data:

The image dataset is divided into two sets: the training data and the test data. The training data is used to train the neural network model, while the test data is used to evaluate the model's performance after training.

3.1.4. Neural Network (Model):

The neural network is the core component of our project. It is a machine learning model inspired by the structure of the human brain. The neural network will learn from the training data to recognize and classify hand gesture images. You mentioned using a convolutional neural network (CNN), which is particularly effective for image-related tasks.

3.1.5. Evaluation:

After training the neural network on the training data, you need to evaluate its performance using the test data. The evaluation process measures how well the model generalizes to unseen data and provides insights into its accuracy and effectiveness.

3.1.6. Image Prediction:

Once the neural network model is trained, it can take new hand gesture images as input and make predictions about which letter from 'A' to 'I' the hand gesture represents.

3.1.7. Flask UI (User Interface):

Flask is a web framework for Python, and in our project, it serves as the user interface (UI). The Flask UI allows users to interact with the trained model through a web application. Users can upload or capture hand gesture images through the UI and receive predictions from the model in response.

3.1.8. User:

The end-users of our project are individuals who want to communicate using hand gestures. They interact with the Flask UI to input their hand gesture images and obtain predictions for the corresponding letters.

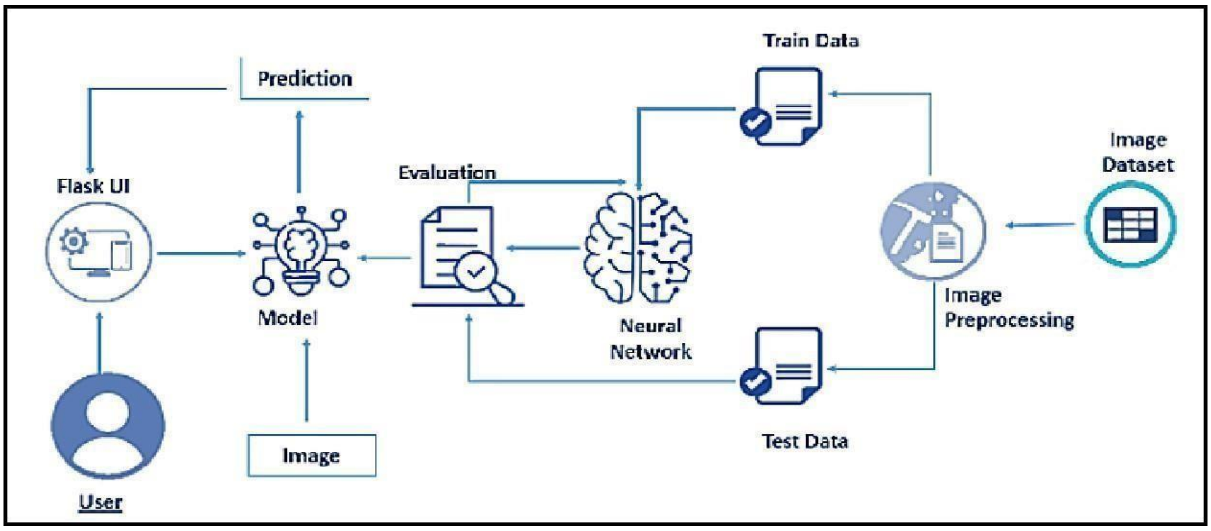


Fig. 3. 1. Block Diagram

3.2 HARDWARE / SOFTWARE REQUIREMENTS

3.2.1 Hardware Requirements:

1.Processor: Intel CoreTM i5-9300H with a clock speed of 2.4GHz.

2.RAM Size: 8GB DDR (Random Access Memory) to handle data and computations efficiently.

3.System Type: X64-based processor architecture for compatibility and performance.

4.Webcam: An integrated or external webcam with Full HD support for capturing high-quality images and video feed.

3.2.2 Software Requirements:

1.Desktop GUI: Anaconda Navigator, which provides an integrated development environment (IDE) for data science tasks and managing Python environments.

2. **Operating System**: Windows 10, as the base operating system for running the project.

3.Front-End: HTML and CSS will be used for developing the user interface and enhancing user experience.

Programming Language:

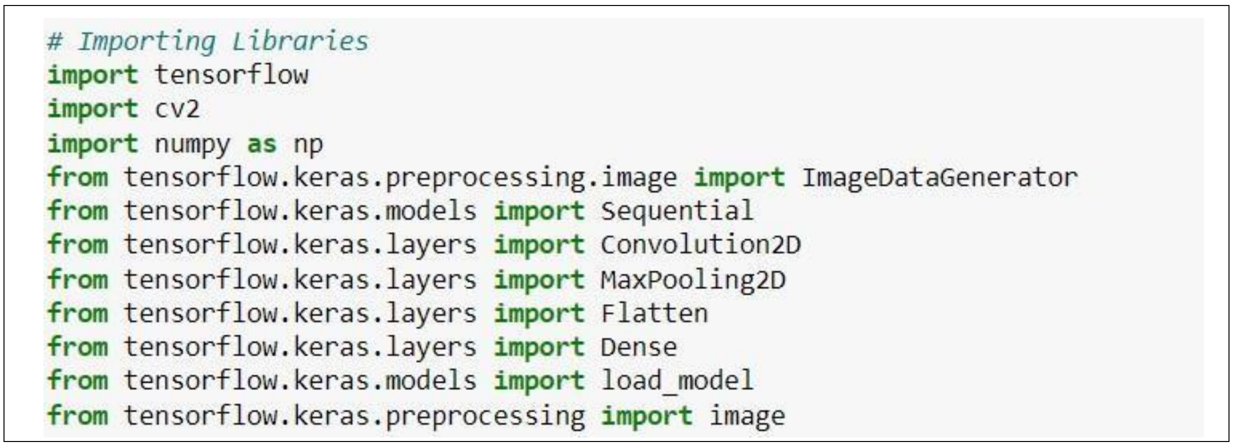
1. Python, which serves as the primary language for implementing the fuel efficiency prediction system using machine learning algorithms.

By utilizing the above hardware and software components, the project can be efficiently developed, offering a user-friendly interface and accurate fuel efficiency predictions for various vehicles.

4 EXPERIMENTAL INVESTIGATIONS

The project involves the development of a recommendation system based on collaborative filtering for book recommendations. The following are the key steps and experimental investigations made during the solution development:

4.1 Importing the Libraries:
In this code snippet, we import the necessary libraries and modules required for building and training a deep learning model for image classification. The TensorFlow library provides the foundation for creating and training the neural network, while OpenCV is used for image processing tasks. Other modules from TensorFlow's Keras API are imported to construct the model architecture and handle image data.



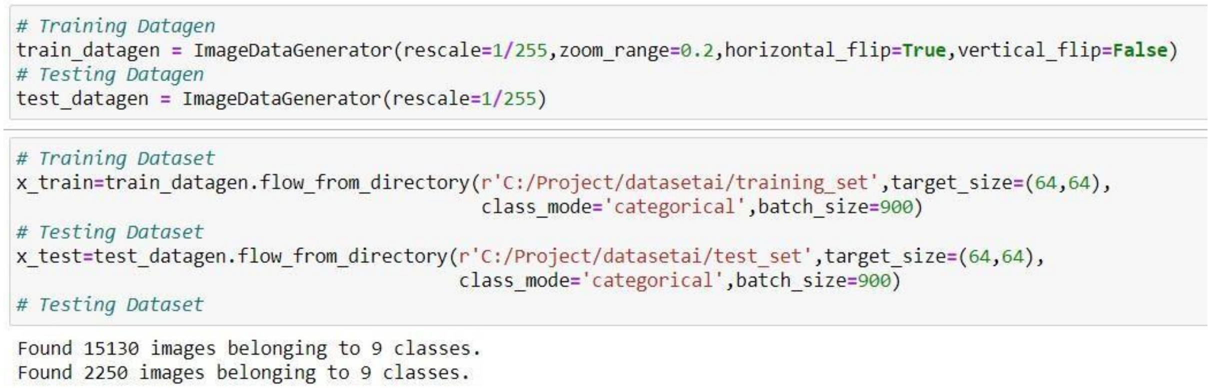
```
# Importing Libraries
import tensorflow
import cv2
import numpy as np
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Convolution2D
from tensorflow.keras.layers import MaxPooling2D
from tensorflow.keras.layers import Flatten
from tensorflow.keras.layers import Dense
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
```

Fig. 4. 1. Importing the libraries

4.2 Data Augmentation:

We modify and augment the images in both the train and test data. The only operation for test data we perform is rescaling the images to 1./255 to. By rescaling images in the test and training set to 1/255 within the ImageDataGenerator, the pixel values are appropriately transformed to meet the requirements for efficient and effective model training. We do not 7 need to augment the test data since we won't use it for training.

We also perform horizontal and vertical flipping to provide robust orientations that take all kinds of directions a hand can be present in front of the system into consideration. Before passing the images to the model, we set shuffle as false in the test when flowing from the directory in order to stop the model from learning wrong labels. We also create a validation set with 20% of the test data in order to test the model how it did each epoch on unseen data.



```
# Training Datagen
train_datagen = ImageDataGenerator(rescale=1/255,zoom_range=0.2,horizontal_flip=True,vertical_flip=False)
# Testing Datagen
test_datagen = ImageDataGenerator(rescale=1/255)
```

```
# Training Dataset
x_train=train_datagen.flow_from_directory(r'C:/Project/datasetai/training_set',target_size=(64,64),
                                          class_mode='categorical',batch_size=900)
# Testing Dataset
x_test=test_datagen.flow_from_directory(r'C:/Project/datasetai/test_set',target_size=(64,64),
                                        class_mode='categorical',batch_size=900)
# Testing Dataset
```

```
Found 15130 images belonging to 9 classes.
Found 2250 images belonging to 9 classes.
```

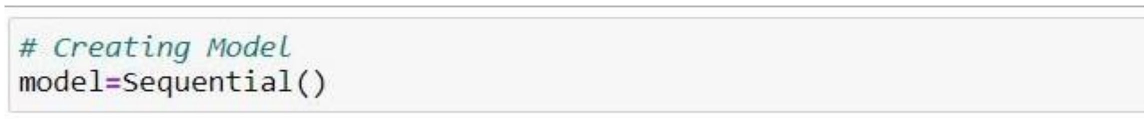
Fig. 4.2 Data Augmentation

4.3 Model Building

In this milestone, we start building our model by:

4.3.1 Initializing the model:

we use the sequential model, which allows us to add layers sequentially one after the other. The model is then built layer by layer to form a deep learning architecture capable of recognizing hand gesture images representing letters from 'A' to 'I'.

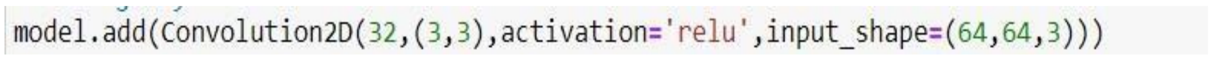


```
# Creating Model
model=Sequential()
```

Fig. 4.3.1 Initializing the model

4.3.2 Adding Convolution layers:

By adding this convolutional layer, the model begins to learn lower-level image features from the input images, paving the way for subsequent layers to learn more abstract and complex features to identify the hand gestures effectively.

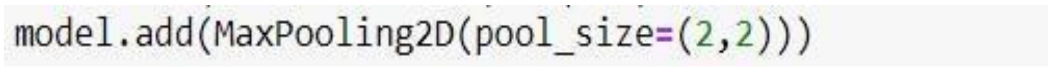


```
model.add(Convolution2D(32,(3,3),activation='relu',input_shape=(64,64,3)))
```

Fig 4.3.2 Adding Convolution Layer

4.3.3 Adding Pooling layers:

Max-pooling helps in achieving spatial invariance and reducing the number of parameters in the model, which can lead to faster computation and reduced overfitting. By adding this maxpooling layer after the convolutional layer, the model learns to capture and retain the most significant features from the input images, enabling more efficient and effective recognition of hand gestures.



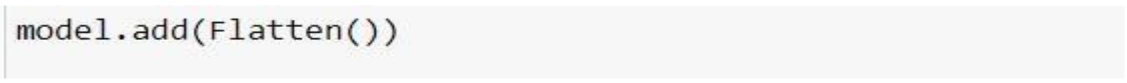
```
model.add(MaxPooling2D(pool_size=(2,2)))
```

Fig 4.3.3 Adding Pooling layers

4.3.4 Flatten layer:

The flatten layer takes the output from the previous layer (in this case, the output from the max-pooling layer) and reshapes it into a 1D vector. This flattening process collapses all the spatial dimensions and preserves the information as a continuous sequence, which can be used as input to the fully connected layers.

By adding the flatten layer, the model prepares the data to be fed into the dense (fully connected) layers for final classification. This allows the model to make predictions based on the learned features from the convolutional and pooling layers, enabling it to recognize hand gestures effectively.

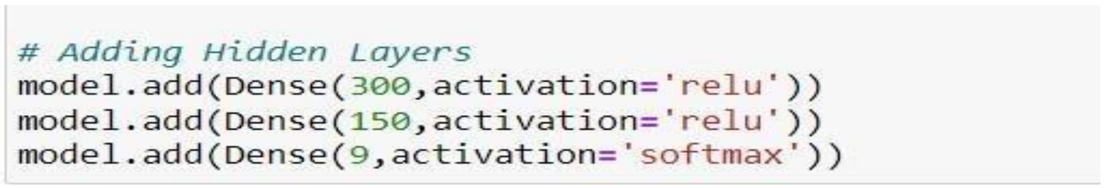


```
model.add(Flatten())
```

Fig 4.3.4 Adding Flatten layers

4.3.5 Hidden layers:

With the addition of hidden layers and the output layer, the neural network is now capable of capturing more complex patterns and making predictions for each class, making it suitable for recognizing hand gestures representing letters 'A' to 'I' in the real-time communication system for deaf and mute individuals.

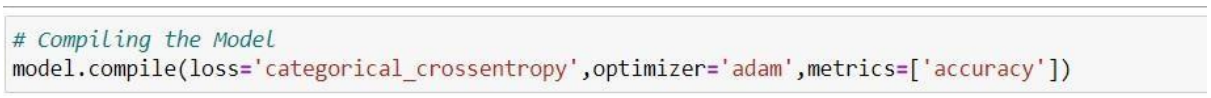


```
# Adding Hidden Layers
model.add(Dense(300,activation='relu'))
model.add(Dense(150,activation='relu'))
model.add(Dense(9,activation='softmax'))
```

Fig.4.3.5 Hidden layers

4.3.6 Compile the Model:

By compiling the model with the appropriate loss function, optimizer, and evaluation metric, we are now ready to start the training process. The model will learn from the augmented and preprocessed training data and gradually improve its accuracy and predictive capabilities during the training epochs.

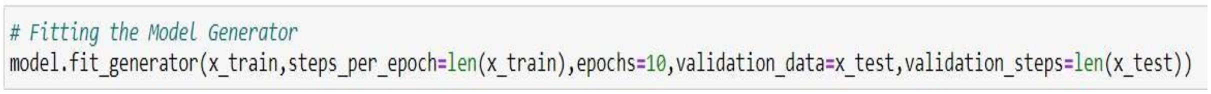


```
# Compiling the Model
model.compile(loss='categorical_crossentropy',optimizer='adam',metrics=['accuracy'])
```

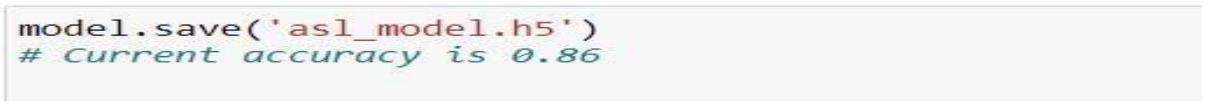
Fig.4.5.6 Compile the model

4.3.4 Fit and Save the Model:

During the training process, the model's performance will be monitored using the categorical cross-entropy loss and accuracy metrics on both the training and validation datasets. After training is complete, the trained model will be saved to the 'asl_model.h5' file, allowing us to use it later for making predictions or further fine-tuning, if necessary.



```
# Fitting the Model Generator
model.fit_generator(x_train,steps_per_epoch=len(x_train),epochs=10,validation_data=x_test,validation_steps=len(x_test))
```



```
model.save('asl_model.h5')
# Current accuracy is 0.86
```

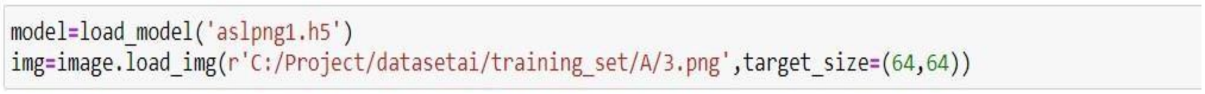
Fig.4. 7.Fit and Save the model

4.4 Test the model:

To test the trained model, we can use the saved model file ('asl_model.h5') and load it back into memory. Once the model is loaded, we can use it to make predictions on new data, which in this case will be the testing dataset.

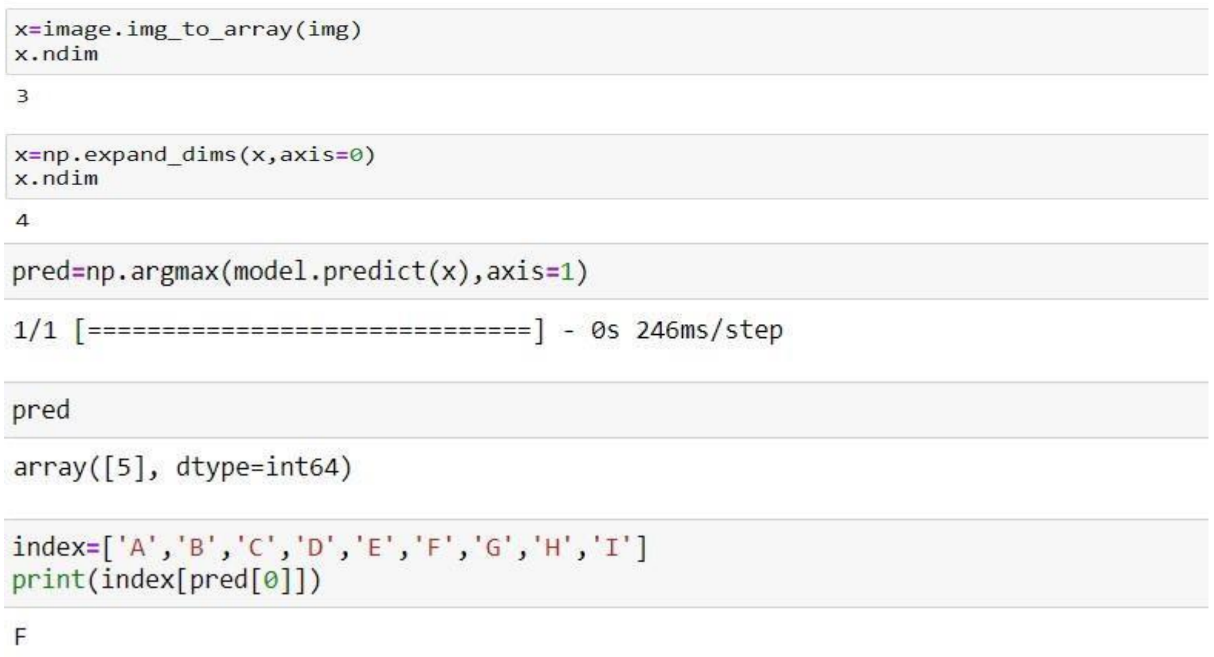
The load_model() function is used to load the model from the 'aslpng1.h5' file into memory. Then, we use the evaluate() function to assess the model's performance on the testing dataset (x_test). This will give us the test loss and test accuracy, which will help us understand how well the model generalizes to unseen data.

By testing the model on the testing dataset, we can determine its real-world performance and ensure that it effectively recognizes hand gestures representing letters 'A' to 'I' in the real-time communication system for deaf and mute individuals.



```
model=load_model('aslpng1.h5')
img=image.load_img(r'C:/Project/datasetai/training_set/A/3.png',target_size=(64,64))
```

```
img
```



```
x=image.img_to_array(img)
x.ndim
```

```
3
```

```
x=np.expand_dims(x,axis=0)
x.ndim
```

```
4
```

```
pred=np.argmax(model.predict(x),axis=1)
```

```
1/1 [==============================] - 0s 246ms/step
```

```
pred
```

```
array([5], dtype=int64)
```

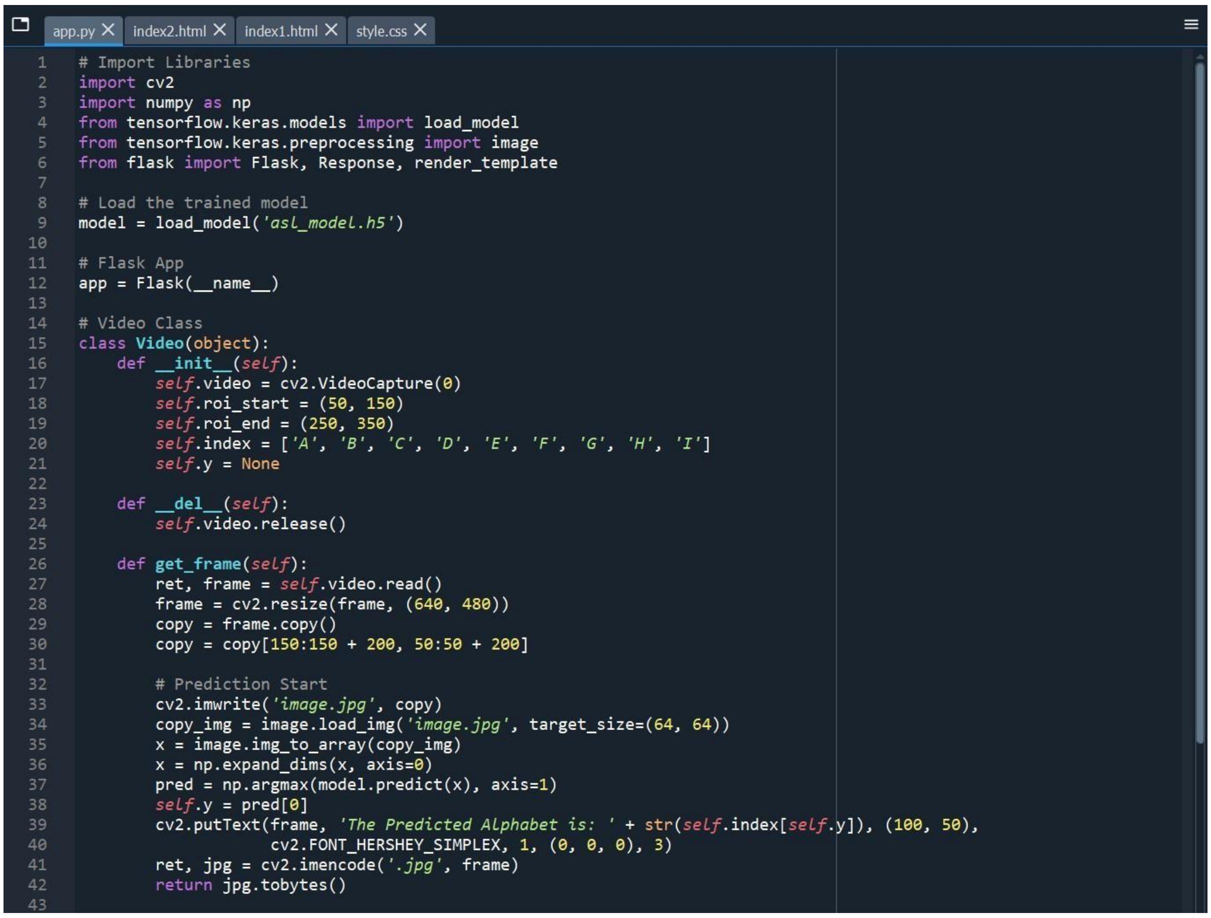
```
index=['A','B','C','D','E','F','G','H','I']
print(index[pred[0]])
```

```
F
```

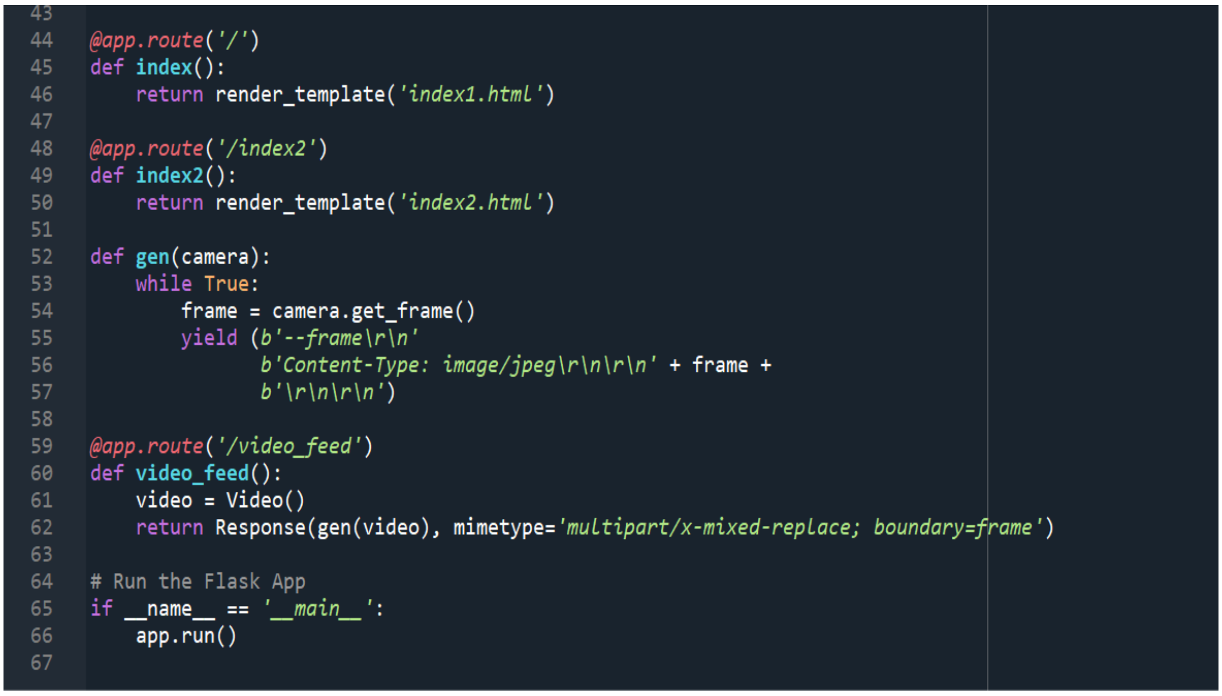
Fig 4.4 Testing the model

4.5 Flask Code:

Python code is written to implement the recommendation system, utilizing the collaborative filtering model. Additionally, Flask, a web application framework, is used to build the frontend of the recommendation system, making it user-friendly and accessible via a web interface.



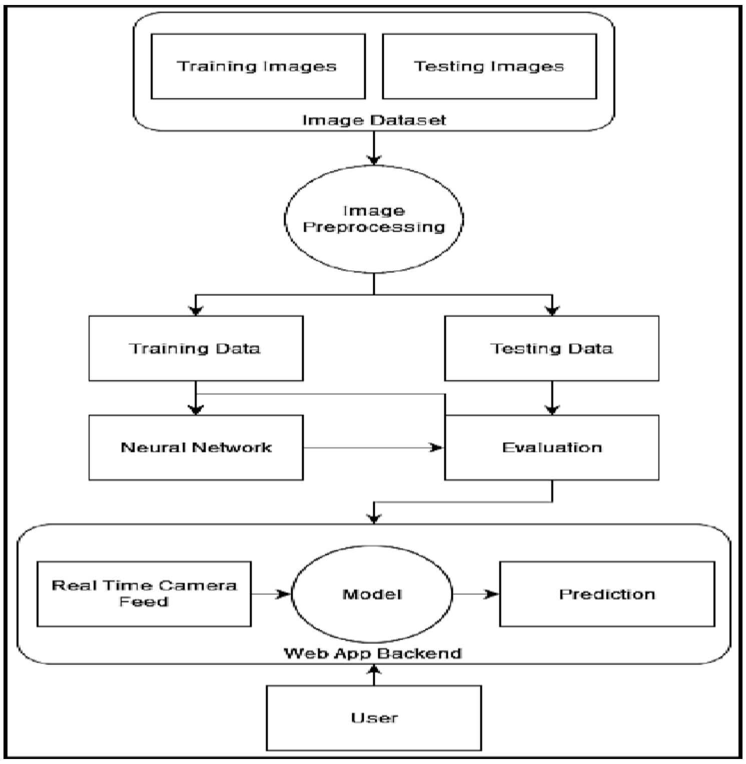
```
# Import Libraries
import cv2
import numpy as np
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from flask import Flask, Response, render_template

# Load the trained model
model = load_model('asl_model.h5')

# Flask App
app = Flask(__name__)

# Video Class
class Video(object):
    def __init__(self):
        self.video = cv2.VideoCapture(0)
        self.roi_start = (50, 150)
        self.roi_end = (250, 350)
        self.index = ['A', 'B', 'C', 'D', 'E', 'F', 'G', 'H', 'I']
        self.y = None

    def __del__(self):
        self.video.release()

    def get_frame(self):
        ret, frame = self.video.read()
        frame = cv2.resize(frame, (640, 480))
        copy = frame.copy()
        copy = copy[150:150 + 200, 50:50 + 200]

        # Prediction Start
        cv2.imwrite('image.jpg', copy)
        copy_img = image.load_img('image.jpg', target_size=(64, 64))
        x = image.img_to_array(copy_img)
        x = np.expand_dims(x, axis=0)
        pred = np.argmax(model.predict(x), axis=1)
        self.y = pred[0]
        cv2.putText(frame, 'The Predicted Alphabet is: ' + str(self.index[self.y]), (100, 50),
                    cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 0), 3)
        ret, jpg = cv2.imencode('.jpg', frame)
        return jpg.tobytes()

```



```python
@app.route('/')
def index():
    return render_template('index1.html')

@app.route('/index2')
def index2():
    return render_template('index2.html')

def gen(camera):
    while True:
        frame = camera.get_frame()
        yield (b'--frame\r\n'
               b'Content-Type: image/jpeg\r\n\r\n' + frame +
               b'\r\n\r\n')

@app.route('/video_feed')
def video_feed():
    video = Video()
    return Response(gen(video), mimetype='multipart/x-mixed-replace; boundary=frame')

# Run the Flask App
if __name__ == '__main__':
    app.run()
```

Fig.4. 8. Flask code

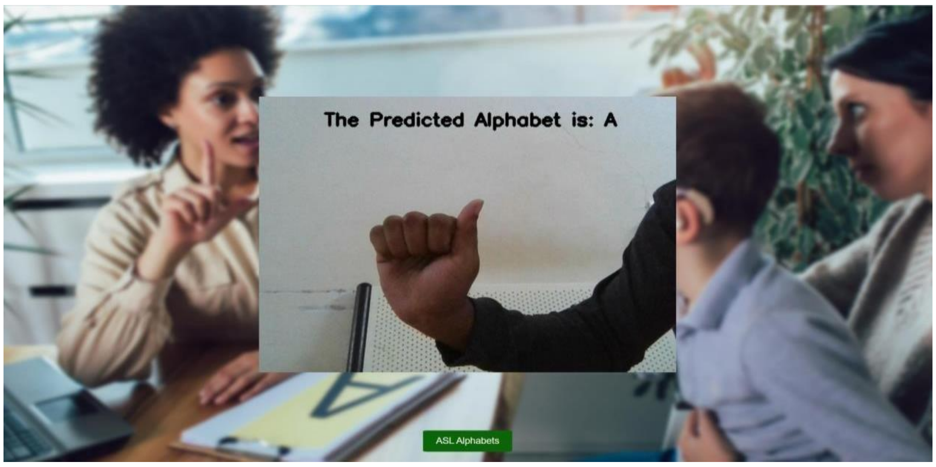
5 FLOWCHART

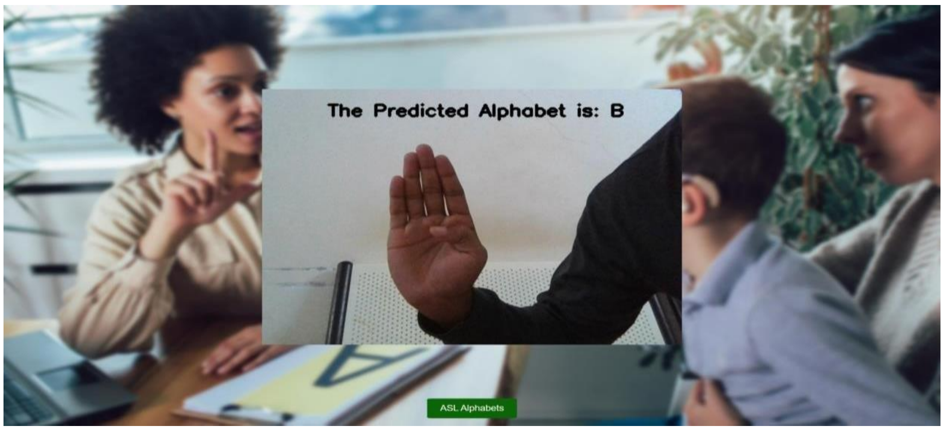


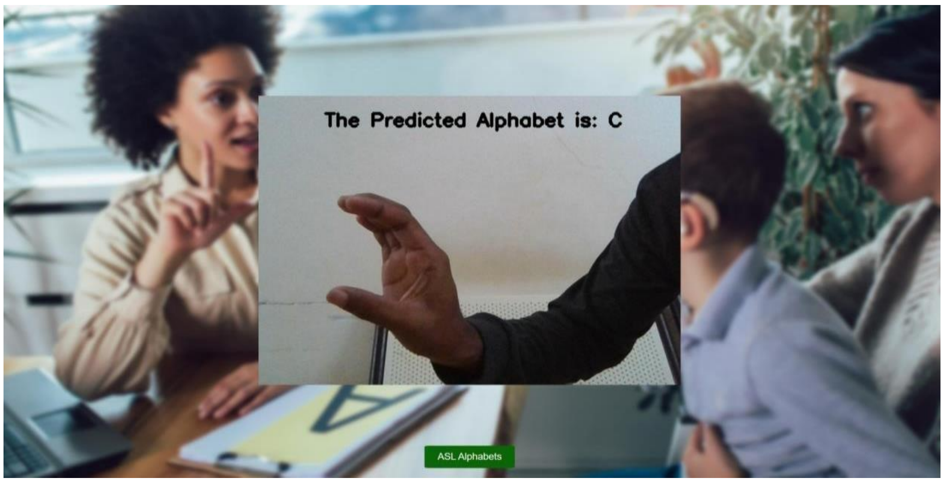
6 RESULT

The proposed procedure was implemented and tested with set of images. The set of 15750 images of Alphabets from “A” to “I” are used for training database and a set of 2250 images of Alphabets from “A” to “I” are used for testing database. Once the gesture is recognise the equivalent Alphabet is shown on the screen.

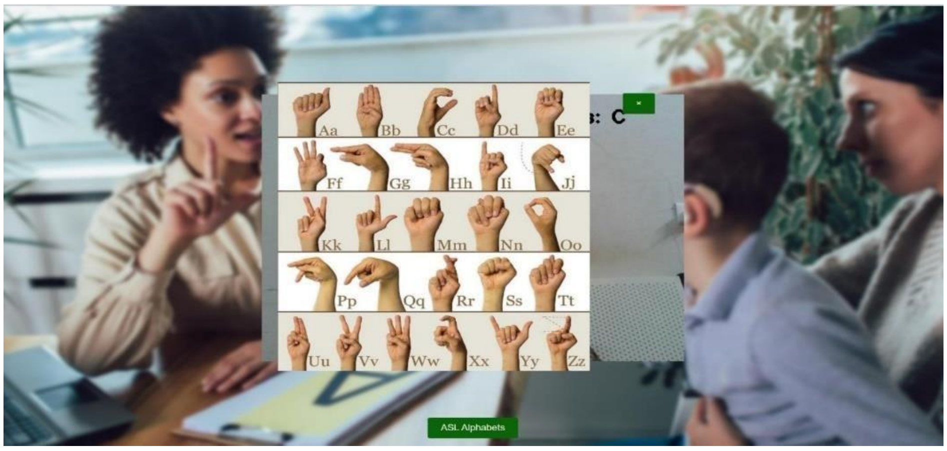
Some sample images of the output are provided below:







Reference of ASL Alphabets:



7.ADVANTAGES AND DISADVANTAGES

7.1 Advantages:

1.Automation: The real-time communication system reduces the need for manual work in understanding hand gestures, enabling efficient and automated communication for deaf and mute individuals.
2.Enhanced Accuracy: The deep learning model used in the system is more accurate than the average human in recognizing hand gestures, leading to more reliable communication outcomes.
3.Scalability: The system can handle large amounts of data, making it suitable for widespread usage and accommodating a diverse range of users.
4.Accessibility: The application can be accessed from any device with an internet connection, making it accessible to users across various platforms.

7.2 Disadvantages:

1.Limited Alphabet Range: The current model only supports recognition of hand gestures representing alphabets from 'A' to 'I'. The extension to include more alphabets requires additional gesture recognition techniques.
2.Gesture Input Limitation: The system may face challenges in recognizing alphabets beyond 'I' as they may require more complex gesture inputs from the user, necessitating further research and development.
3.Low Quantity/Quality of Images: The system's accuracy is affected by the dataset's quantity and quality. Increasing the dataset size and improving image quality could enhance the accuracy of predictions.
4.Occasional Errors: The system may occasionally encounter errors in hand gesture recognition, especially in challenging lighting conditions or noisy environments.

8 APPLICATIONS

1. Smart Home Automation for Accessibility:

Real-time communication systems, when combined with AI, can enable individuals with disabilities to control their home environment. Voice and gesture recognition technologies allow them to operate lights, appliances, doors, and other devices through voice commands or gestures, promoting accessibility within their living spaces.

2. Vision Assistance:

AI-powered real-time communication systems can provide valuable assistance to individuals with visual impairments. By employing advanced algorithms, these systems can identify objects, read text, describe scenes, and offer audio guidance, allowing visually impaired individuals to navigate their surroundings more effectively.

3. Hearing Assistance:

Real-time communication systems with AI capabilities can enhance communication for individuals with hearing impairments. Speech recognition and transcription algorithms can convert spoken words into text, enabling individuals to read conversations in real-time or through visual displays.

9 CONCLUSION

This project successfully demonstrates a web application that harnesses the power of deep learning, computer vision, and neural networks to recognize American Sign Language (ASL). Developed using Flask and Python, the application achieves impressive results, with a training accuracy of 98% and a testing accuracy of approximately 99%.

The proposed system offers scalability, capable of handling a large number of users without compromising performance. Being a web application, it enjoys compatibility with various devices that can run a web browser, making it easily accessible to users across platforms.

The real-world applications of this project are diverse and impactful. Apart from facilitating communication for deaf and mute individuals, it can find use in recognizing numbers, processing bank cheques, and other practical scenarios. Its potential for integration into various domains makes it a valuable addition to assistive technologies.

The project's success in bridging the communication gap between deaf individuals and the rest of society is commendable. By allowing two-way communication through sign language, it empowers deaf individuals to express themselves effectively and fosters inclusive interactions in society.

10 FUTURE SCOPE

Extended Alphabet Recognition: With the introduction of gesture recognition, the web application can be easily expanded to recognize letters beyond 'I'. By incorporating more sign language symbols and gestures, the system can support a broader range of communication, allowing users to express themselves more comprehensively.

Multi-Digit Recognition: Building upon the existing capabilities, the application can be further developed to recognize multiple digits. This enhancement would enable users to communicate numbers and perform arithmetic operations using sign language, expanding the system's utility in various contexts.

Symbol Recognition: Expanding the system to recognize other symbols and expressions used in sign language would enable users to convey complex messages, emotions, and concepts more effectively.

Interface Control: Leveraging gesture recognition, the web application can go beyond language recognition and enable users to control software and hardware interfaces. This feature could empower specially-abled individuals to interact with technology and devices using intuitive hand gestures.

Real-Time Translation: Implementing real-time translation features would allow the application to convert sign language into spoken language or text, facilitating seamless communication between deaf individuals and the general public.

Customization and Personalization: Incorporating customization options would enable users to adapt the application to their specific sign language preferences and needs, making it more user-friendly and tailored to individual requirements.

Mobile Integration: Developing a mobile version of the application would make it accessible on smartphones and tablets, allowing users to communicate on-the-go and fostering independence and convenience.

Collaboration with Experts: Collaborating with sign language experts, linguists, and the deaf community can provide valuable insights and feedback for improving the system's accuracy and cultural relevance.

11.APPENDIX

Source code

This the source code link given below:

Code Link