GAUTAM BUDDHA UNIVERSITY

MAJOR PROJECT REPORT ON

SIGN LANGUAGE RECOGNITION

SCHOOL OF INFORMATION AND COMMUNICATION TECHNOLOGY

UNDER THE GUIDANCE OF

Mr. Neeraj Kaushik

MCA(AI) SEM IV [2023-2024]

Submitted By-

Group Member	Roll Number
Rajkumar	225/PCA/011
Shabnam Kumari	225/PCA/027
Soni Saraswat	225/PCA/030
Akanksha Tyagi	225/PCA/026

SCHOOL OF INFORMATION AND COMMUNICATION
TECHNOLOGY GAUTAM BUDDHA UNIVERSITY, GREATER NOIDA,
201312, U.P., (INDIA)

Candidate's Declaration

We, hereby, certify that the work embodied in this minor project report entitled "**SIGN LANGUAGE RECOGNITION**" in partial fulfillment of the requirements for the award of the Degree of MCA submitted to the School of Information and Communication Technology, Gautam Buddha University, Greater Noida is an authentic record of our own work carried out under the supervision of Mr. Neeraj Kaushik , School of ICT. The matter presented in this report has not been submitted in any other University / Institute for the award of any other degree or diploma. Responsibility forany plagiarism related issue stands solely with us:

Group Member	Roll Number
Rajkumar	225/PCA/011
Shabnam Sharma	225/PCA/027
Soni Saraswat	225/PCA/030
Akanksha Tyagi	225/PCA/026

This is to certify that the above statement made by the candidates is correct to the best of my knowledge and belief. However, responsibility for any plagiarism related issue solely stands with the students.

Signature of the Supervisor:

Name: Mr. Neeraj Kaushik

Date /Place: 05/March/2024, Greater Noida

Acknowledgement

We have taken efforts in this project. However, it would not have been possible without the kind support and help of many individuals and organization. We would like to extend my sincere thanks to all of them.

We are highly indebted to **Mr. Neeraj Kaushik** for his guidance and constant supervision as well as for providing necessary information regarding the project & also for his support in completing the project.

We would like to thank Dean of ICT Prof. **Sanjay kumar sharma** and HOD of ICT,

Dr. Neeta Singh whose support helped us to complete our work. We would like to express our gratitude towards our friends & family members for their kind cooperation and encouragement which helped us in completion of this major project.

Table of Contents

1. List of figures

2. Abstract

Sign Language Recognition Using Random Forest

Sign language serves as a vital communication medium for the deaf and hard of hearing community. In this project, we focus on recognizing sign language gestures using state-of-the-art machine learning techniques, with a specific emphasis on the Random Forest algorithm.

Key Points:

Background and Motivation:
Sign language bridges communication gaps for individuals with hearing impairments.
Existing study materials for sign language learning are limited, making the learning process challenging.
Our project aims to address this gap by leveraging machine learning to recognize sign gestures.

Dataset Collection and Feature Extraction:
We collect a comprehensive dataset of sign language gestures.
Various feature extraction techniques are applied to extract meaningful information from the data.

Random Forest Algorithm:
Random Forest is an ensemble learning method that combines multiple decision trees.
During training, it constructs several decision trees using random subsets of the dataset and features.
The aggregation of results from individual trees improves accuracy and reduces overfitting.

Cross-Validation and Evaluation:
We perform four-fold cross-validation to assess the performance of our approach.
The validation set consists of images from different individuals not present in the training set.

Results and Impact:
Our Random Forest-based model achieves accurate sign language recognition.
By avoiding the need for expensive external sensors, we enhance accessibility for learners.
This work contributes to creating a more inclusive environment for the deaf community.
In conclusion, our project demonstrates the effectiveness of Random Forest in sign language recognition, emphasizing affordability and real-time applicability. By advancing this field, we hope to empower individuals to communicate seamlessly through sign language.

3. Introduction

The goal of this project was to build a neural network able to classify which letter of the American Sign Language (ASL) alphabet is being signed, given an image of a signing hand. This project is a first step towards building a possible sign language translator, which can take communications in sign language and translate them into written and oral language. Such a translator would greatly lower the barrier for many deaf and mute individuals to be able to better communicate with others in day to day interactions.

This goal is further motivated by the isolation that is felt within the deaf community. Loneliness and depression exists in higher rates among the deaf population, especially when they are immersed in a hearing world [1]. Large barriers that profoundly affect life quality stem from the communication disconnect between the deaf and the hearing. Some examples are information deprivation, limitation of social connections, and difficulty integrating in society [2].

Most research implementations for this task have used depth maps generated by depth camera and high resolution images. The objective of this project was to see if neural networks are able to classify signed ASL letters using simple images of hands taken with a personal device such as a laptop webcam. This is in alignment with the motivation as this would make a future implementation of a real time ASL-to-oral/written language translator practical in an everyday situation.

4. Description of Overall Software Structure

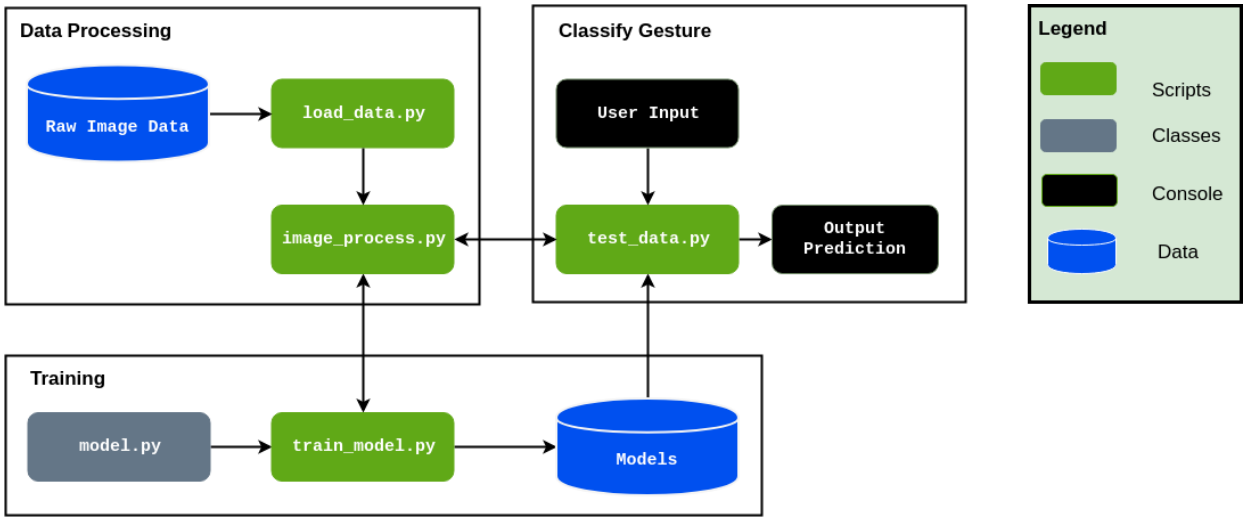


Figure 1: Block Diagram of Software

As shown in Figure 1, the project will be structured into 3 distinct functional blocks, **Data Processing**, **Training**, **Classify Gesture**. The block diagram is simplified in detail to abstract some of the minutiae:

4.1 Data Processing: The load data.py script contains functions to load the Raw Image Data and save the image data as numpy arrays into file storage. The process data.py script will load the image data from data.npy and preprocess the image by resizing/rescaling the image, and applying filters and ZCA whitening to enhance features. During training the processed image data was split into training, validation, and testing data and written to storage. Training also involves a load dataset.py script that loads the relevant data split into a Dataset class. For use of the trained model in classifying gestures, an individual image is loaded and processed from the filesystem.

4.2 Training: The training loop for the model is contained in train model.py. The model is trained with hyperparameters obtained from a config file that lists the learning rate, batch size, image filtering, and number of epochs. The configuration used to train the model is saved along with the model architecture for future evaluation and tweaking for improved results. Within the training loop, the training and validation datasets are loaded as Dataloaders and the model is trained using Adam Optimizer with Cross Entropy Loss. The model is evaluated every epoch on the validation set and the model with best validation accuracy is saved to storage for further evaluation and use. Upon finishing training, the training and validation error and loss is saved to the disk, along with a plot of error and loss over training.

4.3 Classify Gesture: After a model has been trained, it can be used to classify a new ASL gesture that is available as a file on the filesystem. The user inpūts the filepath of the gesture image and the test data.py script will pass the filepath to process data.py to load and preprocess the file the same way as the model has been trained.

5. Sources of Data

5.1 Data Collection

The primary source of data for this project was the compiled dataset of American Sign Language (ASL) called the ASL Alphabet from Kaggle user Akash [3]. The dataset is comprised of 87,000 images which are 200x200 pixels. There are 29 total classes, each with 3000 images, 26 for the letters A-Z and 3 for space, delete and nothing. This data is solely of the user Akash gesturing in ASL, with the images taken from his laptop's webcam. These photos were then cropped, rescaled, and labelled for use.

(a.) ASL letter A (b.) ASL letter E (c.) ASL letter H (d.) ASL letter Y

Figure 2: Examples of images from the Kaggle dataset used for training. Note difficulty of distinguishing fingers in the letter E.

A self-generated test set was created in order to to investigate the neural network's ability to generalize. Five different test sets of images were taken with a webcam under different lighting conditions, backgrounds, and use of dominant/non-dominant hand. These images were then cropped and preprocessed.

5.2 Data Pre-processing

The data preprocessing was done using the PILLOW library, an image processing library, and sklearn.decomposition library, which is useful for its matrix optimization and decomposition functionality.

5.2.1 Sample from non-uniform background test set

5.2.2 Sample from plain white background test set

5.2.3 Sample from a darker test set

5.2.4 Sample from plain white background test set

Figure 3: Examples of the signed letter A T from two test sets with differing lighting and background

Image Enhancement: A combination of brightness, contrast, sharpness, and color enhancement was used on the images. For example, the contrast and brightness were changed such that fingers could be distinguished when the image was very dark.

Edge Enhancement: Edge enhancement is an image filtering techniques that makes edges more defined. This is achieved by the increase of contrast in a local region of the image that is detected as an edge. This has the effect of making the border of the hand and fingers, versus the background, much more clear and distinct. This can potentially help the neural network identify the hand and its boundaries.

Image Whitening: ZCA, or image whitening, is a technique that uses the singular value decomposition of a matrix. This algorithm decor relates the data, and removes the redundant, or obvious, information out of the data. This allows for the neural network to look for more complex and sophisticated relationships, and to uncover the underlying structure of the patterns it is being trained on. The covariance matrix of the image is set to identity, and the mean to zero.

Figure 4: Examples of image preprocessing

6. Machine Learning Model

MODEL TRAINING

Random forest algorithm is used to train the machine learning model.

Random forest Algorithm

Forest is known for its ability to handle noisy data and its ability to estimate feature importance, making it a popular algorithm for many realworld applications. Random Forest is an ensemble machine learning algorithm that operates by constructing a multitude of decision trees and aggregating their predictions.

It is used for both regression and classification problems. The code is created for Random Forest Classifier algorithm using the scikit-learn library. It sets the number of trees in the forest to 30 using the "n_estimators" parameter. The ".fit" method trains the classifier on the training data "X_train" and corresponding target values "y_train".The ".predict" method is then used to generate predictions for the test data "X_test".Finally, the code prints the accuracy of the predictions using the "metrics.accuracy_score" function from scikit-learn, which compares the actual target values "y_test" to the predicted values "y_pred". The accuracy score is the proportion of correct predictions in the test data.

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of combining multiple classifiers to solve a complex problem and to improve the performance of the model.

As the name suggests, **"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.

The below diagram explains the working of the Random Forest algorithm:

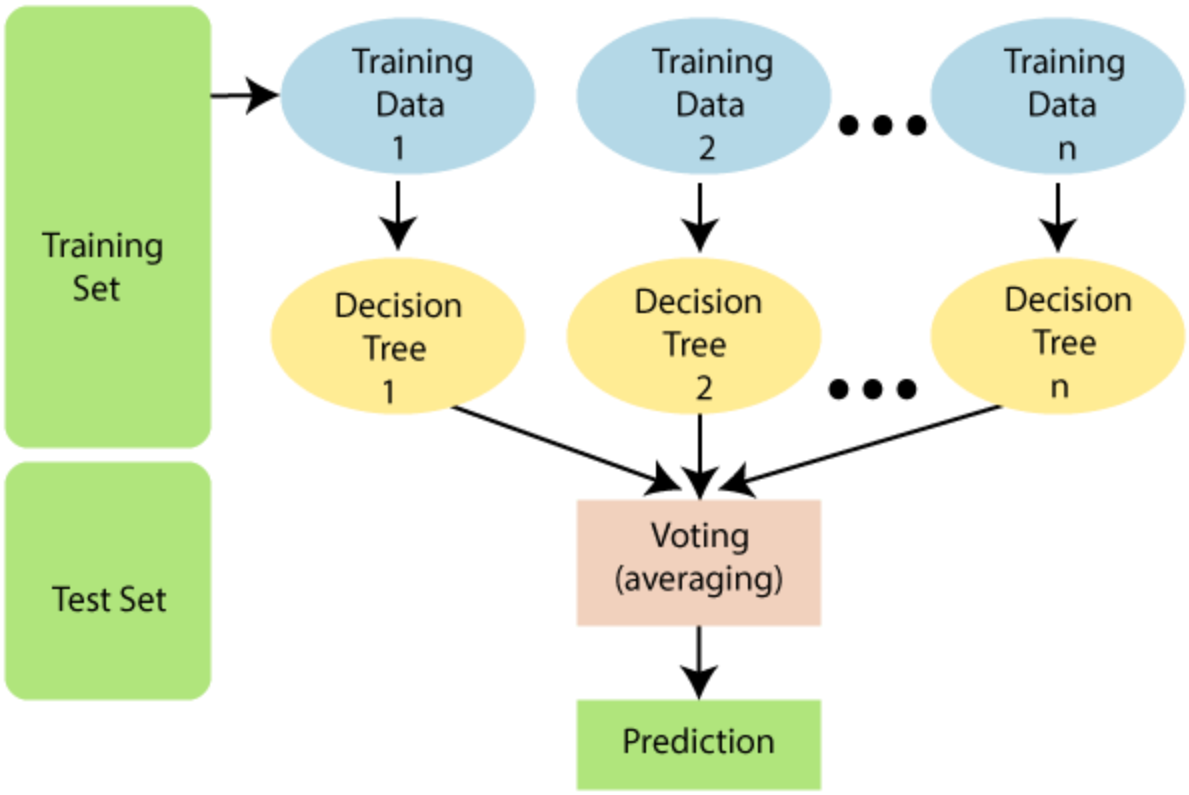


Figure 5: Explains the working of the Random Forest algorithm

7. MODEL TESTING

Model testing is an essential step in evaluating the performance of machine learning models. In the context of sign language recognition using random forest, k-nearest neighbors (KNN), and support vector machine (SVM), the following steps can be taken for model testing: Model Evaluation: Evaluate the performance of the models using the testing set. Common performance metrics for classification tasks include accuracy, precision, recall, and F1 score. It is also essential to examine the confusion matrix to identify which classes are often confused by the models.

1. Confusion matrix
2. Accuracy
3. Precision
4. Recall

For simplicity, we will mostly discuss things in terms of a binary classification problem where let's say we'll have to find if an image is of a cat or a dog. Or a patient is having cancer (positive) or is found healthy (negative). Some common terms to be clear with are:
True positives (TP): Predicted positive and are actually positive.
False positives (FP): Predicted positive and are actually negative.
True negatives (TN): Predicted negative and are actually negative.
False negatives (FN): Predicted negative and are actually positive.

OUTPUT PREDICTION

Sign language recognition module: A module that recognizes the sign language gestures performed by the user and provides feedback on accuracy.

Figure 6: Annotated landmarks

Mediapipe hands object as input and returns a dictionary containing the distance data and the original image with annotated landmarks.

1. The function first converts the image to RGB and processes it with the Mediapipe hands object.
2. It then extracts the hand landmark data from the Mediapipe output and calculates the distances between various landmarks on the hand.
3. The resulting distance data is stored in a list and returned along with the annotated image.

8. Software and hardware requirements

The purpose of system requirement specification is to produce the specification analysis of the task and also to establish complete information about the requirement, behavior and other constraints such as functional performance and so on. The goal of system requirement specification is to completely specify the technical requirements for the product in a concise and unambiguous manner.

Hardware Requirements

- **Processor** - Intel i3 above
- **Speed** - 3.19 GHZ
- **RAM** - minimum 4GB
- **SSD** - 10 GB above Software Requirements
- **Operating System** - Windows 10
- **Front End** - Tkinter
- **Back End** – PYTHON
- **Tool** - Anaconda ,VS code

9 . RESULT

In this section, analyze the results of the proposed system. The screenshots are the results of the system.

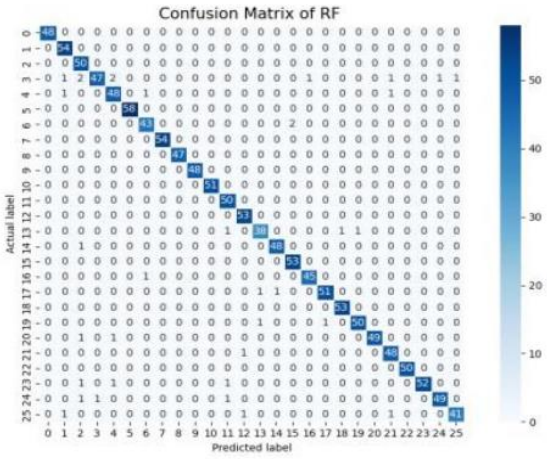
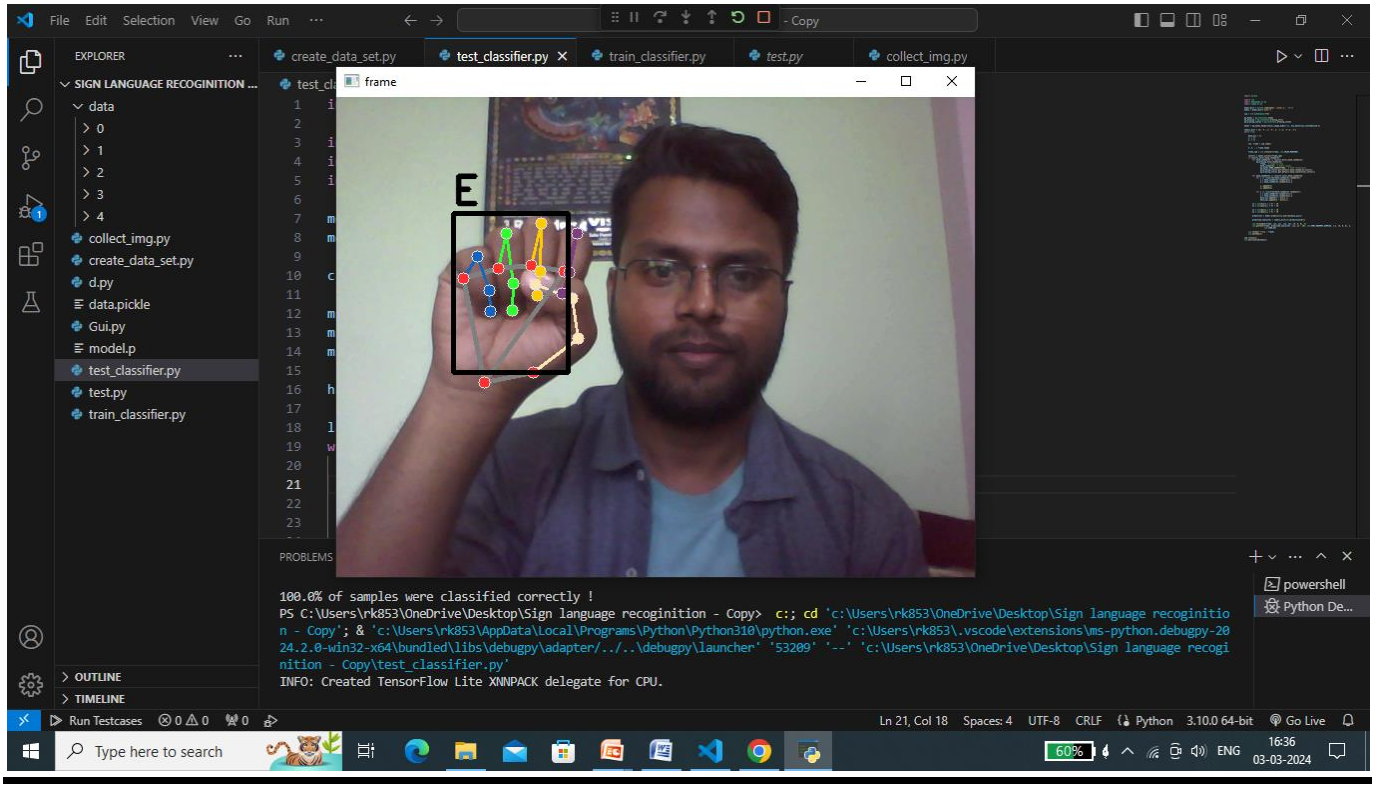
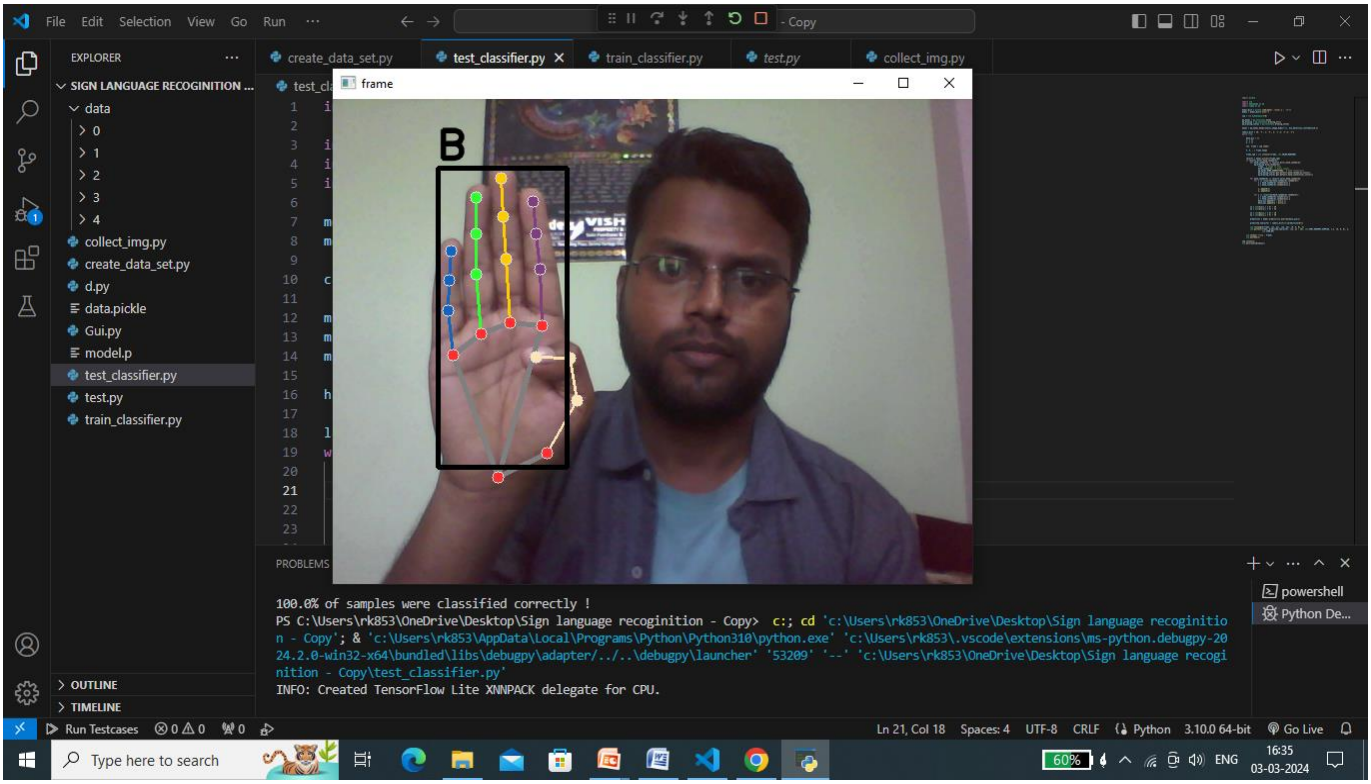
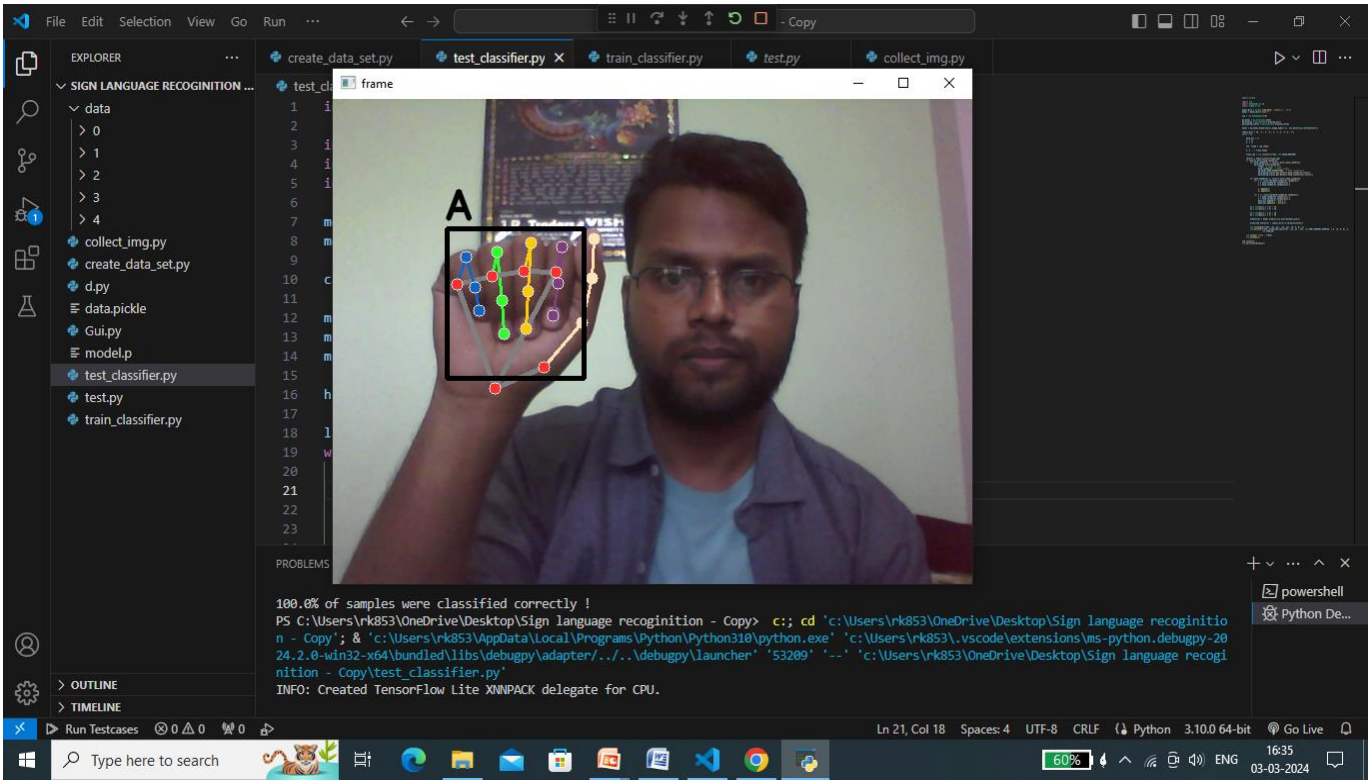


Figure 7: Confusion matrix of Random forest




frame
B
100.0% of samples were classified correctly !
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.


frame
A
100.0% of samples were classified correctly !
INFO: Created TensorFlow Lite XNNPACK delegate for CPU.

10. FUTURE WORK

Expanded Sign Language Vocabulary: Currently, the system may be designed to recognize a limited set of sign language gestures or individual letters. A future enhancement could involve expanding the vocabulary to include more complex phrases or words, allowing for more comprehensive communication. Real-Time Translate: Integrating real-time translation capabilities can be a valuable addition to the system. This would involve converting the recognized sign language gestures into text or spoken language, facilitating communication between sign language users and non-sign language speakers. Improved Gesture Recognition: Enhancing the accuracy and robustness of gesture recognition is an ongoing area of research. This can involve exploring advanced machine learning techniques, such as deep learning, and leveraging larger and more diverse datasets to train the models. Multi-modal Input: Currently, the system may rely on video input from a webcam to detect sign language gestures. However, incorporating other input modalities, such as depth sensors or wearable devices, can provide additional information and improve the system's performance and accuracy. Adaptability to User Preferences: Allowing users to customize and adapt the system to their specific sign language dialect or preferences can be valuable. This may involve incorporating user profiles and preferences into the system to personalize the recognition process.

11. Code

1. Image collection:

```
import os
import cv2


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

number_of_classes = 3
dataset_size = 100

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

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

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

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

        counter += 1

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

2. Create data set:

```
import os
import pickle

import mediapipe as mp
import cv2
import matplotlib.pyplot as plt

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

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

DATA_DIR = './data'

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

        x_ = []
        y_ = []

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

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

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

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

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

3. Train classifier

```
import pickle

from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
import numpy as np
data_dict = pickle.load(open('./data.pickle', 'rb'))

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

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

model = RandomForestClassifier()

model.fit(x_train, y_train)

y_predict = model.predict(x_test)

score = accuracy_score(y_predict, y_test)

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

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

4. Test Classifier

```
import pickle

import cv2
import mediapipe as mp
import numpy as np

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

cap = cv2.VideoCapture(0)

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

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

labels_dict = {0: 'A', 1: 'B', 2: 'C',}
while True:

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

    ret, frame = cap.read()

    H, W, _ = frame.shape

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

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

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

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

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

    x1 = int(min(x_) * W) - 10
    y1 = int(min(y_) * H) - 10

    x2 = int(max(x_) * W) - 10
    y2 = int(max(y_) * H) - 10

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

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

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

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

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

12. Conclusion

Sign language recognition is a crucial field that bridges communication gaps between hearing-impaired individuals and the rest of the world. In this context, employing machine learning algorithms can significantly enhance the accuracy and efficiency of sign language interpretation.

Here are the key takeaways from our exploration of sign language recognition using Random Forest:

Random forest

Random Forest Algorithm:
Powerful Tree Learning Technique: Random Forest is a robust ensemble learning model that combines multiple decision trees to provide a single output.
Training Phase: During training, it creates several decision trees, each constructed using a random subset of the dataset and a random subset of features.
Variability and Over fitting: The randomness introduced in constructing individual trees helps reduce over fitting and improves overall prediction performance.
Aggregation: In prediction, the algorithm aggregates results from all trees, either by voting (for classification tasks) or averaging (for regression tasks).
Stability and Precision: The collaborative decision-making process yields stable and precise results.

Ensemble learning

Ensemble Learning Models:
Strength in Diversity: Ensemble models, like Random Forest, work similarly to a group of diverse experts collaborating to make decisions.
Collective Wisdom: By combining different models (often of the same type or different types), ensemble learning leverages collective wisdom to overcome individual limitations.
Other Ensemble Models: Besides Random Forest, popular ensemble models include XGBoost, AdaBoost, LightGBM, Bagging, and Voting.

Application to Sign Language Recognition:

High Recognition Efficiency:
Random Forest has demonstrated high recognition efficiency in sign language recognition tasks.

Real-Time Efficiency:

While Extreme Gradient Boosting and Support Vector Machine also perform well, Random Forest strikes a balance between accuracy and real-time efficiency.

Handling Complex Data:
Random Forest excels in handling complex data, making it suitable for sign language recognition scenarios.

In summary, the Random Forest algorithm, with its collaborative decision-making process and stability, holds promise for accurate and efficient sign language recognition. Researchers and practitioners can continue exploring this approach to improve communication accessibility for the hearing-impaired community.

13. REFERENCE

1. Bantupalli, K., & Xie, Y. (2018). *American Sign Language Recognition using Deep Learning and Computer Vision. 2018 IEEE International Conference on Big Data (Big Data).*
2. A. Tharwat, T. Gaber, A. E. Hassanien, M. K. Shahin, and B. Refaat, ‘‘Sift-based arabic sign language recognition system,’’ in Proc. AfroEur. Conf. Ind. Advancement. Cham, Switzerland: Springer, 2015, pp. 359–370.
3.] R. Cui, H. Liu, and C. Zhang, ‘‘A deep neural framework for continuous sign language recognition by iterative training,’’ IEEE Trans. Multimedia, vol. 21, no. 7, pp. 1880–1891, Jul. 2019.
4. P. S. Santhalingam, P. Pathak, J. Košecká, and H. Rangwala, ‘‘Sign language recognition analysis using multimodal data,’’ in Proc. IEEE Int. Conf. Data Sci. Adv. Anal. (DSAA), Oct. 2019, pp. 203–210.
5. A. C. Duarte, ‘‘Cross-modal neural sign language translation,’’ in Proc. 27th ACM Int. Conf. Multimedia, Oct. 2019, pp. 1650–1654.
6. M. J. Cheok, Z. Omar, and M. H. Jaward, ‘‘A review of hand gesture and sign language recognition techniques,’’ Int. J. Mach. Learn. Cybern., vol. 10, no. 1, pp. 131–153, Jan. 2019.
7. Q. Xiao, Y. Zhao, and W. Huan, ‘‘Multi-sensor data fusion for sign language recognition based on dynamic Bayesian network and convolutional neural network,’’ Multimedia Tools Appl., vol. 78, no. 11, pp. 15335–15352, Jun. 2019.
8. E. K. Kumar, P. V. V. Kishore, M. T. K. Kumar, and D. A. Kumar, ‘‘3D sign language recognition with joint distance and angular coded color topographical descriptor on a 2–stream CNN,’’ Neurocomputing, vol. 372, pp. 40–54, Jan. 2020.
9. M. J. Cheok, Z. Omar, and M. H. Jaward, ‘‘A review of hand gesture and sign language recognition techniques,’’ Int. J. Mach. Learn. Cybern., vol. 10, no. 1, pp. 131–153, Jan. 2019.
10. Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton, and P. Presti, ‘‘American sign language recognition with the kinect,’’ in Proc. 13th Int. Conf. Multimodal Interfaces (ICMI), 2011, pp. 279–286.
11. Wikipedia
12. Youtube
13. Keggle