PROJECT TITLE:

Sign Language Detection

Team Members:

Pasala Jahnavi-20MID0094
Nagasarala Devisree-20MID0085
Methuku Rithvika Radhe-20MID0073
Tagalapally Varadh Kumar-20MID0153

Campus-VIT VELLORE

GitHub link :

https://github.com/Jahnavi05012003/Sign_language_detection

INTRODUCTION

Sign language is a visual mode of communication used by individuals who are deaf or hard of hearing. Signers use hand gestures, body movements, and facial expressions to convey meaning. Sign-to-text conversion systems aim to automatically translate sign language gestures into text, enabling communication between signers and non-signers. Gaussian blur filters can be used as a pre-processing step in sign-to-text conversion systems. Gaussian blur is a type of image filtering that applies a blurring effect to an image, effectively reducing noise and enhancing image features. Gaussian blur can help to smooth out the image, reducing pixel-level noise that may arise from imperfect camera quality or hand movement during sign language gestures. This can help improve the accuracy of gesture recognition in sign-to-text conversion systems by reducing the impact of noise and improving the robustness of the model.

LITERATURE SURVEY

1. **Indian Sign Language Recognition using SVM Classifier**
   Deepali G. Malia Nitin S.Limkarb Satish H. Malic

After collecting the database from user we need to preprocess those images. Preprocessing images commonly used for removing low-frequency background noise, normalizing the intensity of the individual particles images, Firstly we convert RGB images into grey scale images by using MATLAB (rgb to gray converter). This will convert RGB images to high intensity Grey scale images. In this step we can perform segmentation and noise removal operation. The main aim of pre- processing is an improvement in input data (sign language images) that data suppresses unwanted distortions. Image preprocessing technique uses the considerable redundancy in images. Neighboring pixel corresponding to one object in real image have adjusted some or similar brightness value.. SVM is a support vector machine used for supervised learning model with associated algorithm that analyzed data used for classification and analysis by using Extracted features. SVM classifier is the method of performing the classification task.

2. **A Review Paper on Sign Language Recognition for The Deaf and Dumb**
   Authors: R Rumana, Reddygari Sandhya Rani , Mrs. R. Prema

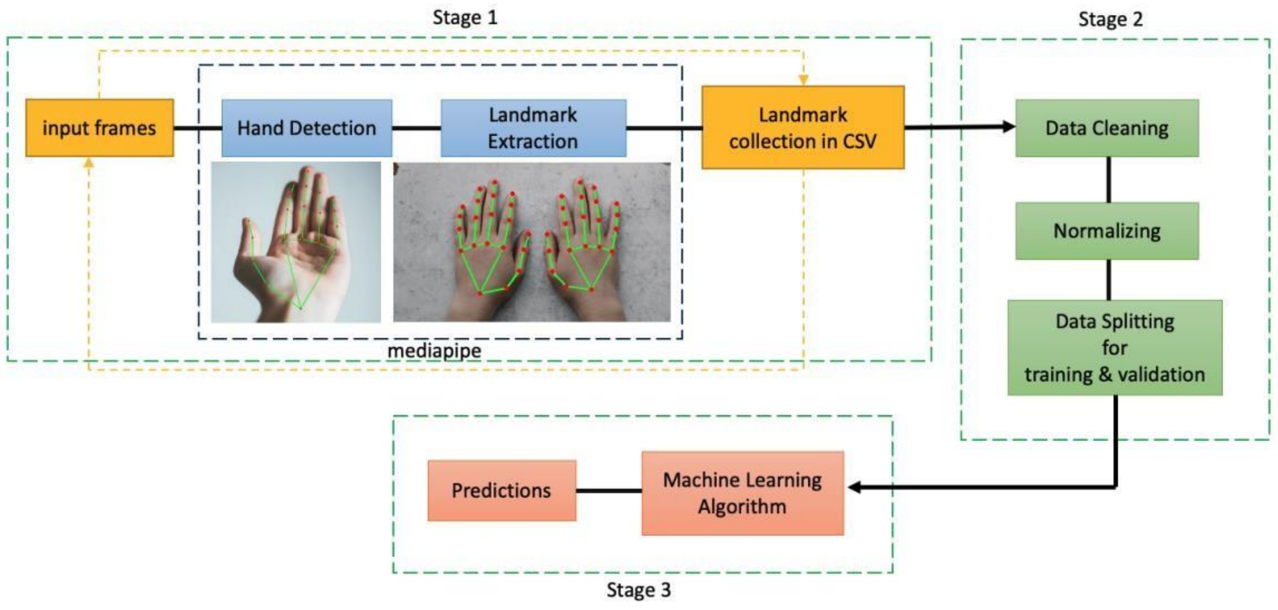
In this paper a method for automatic recognition of signs on the basis of shape-based features is presented. For segmentation of hand region from the images, Otsu’s thresholding algorithm is used, that chooses an optimal threshold to minimize the within-class variance of threshold black and white pixels. Paper presented the recent research and development of sign language based on manual communication and body language.Sign language recognition system typically elaborate three steps preprocessing, feature extraction and classification. Classification methods used for recognition are Neural Network (NN), Support Vector Machine (SVM), Hidden Markov Models (HMM), Scale Invariant FeatureTransform (SIFT), etc.

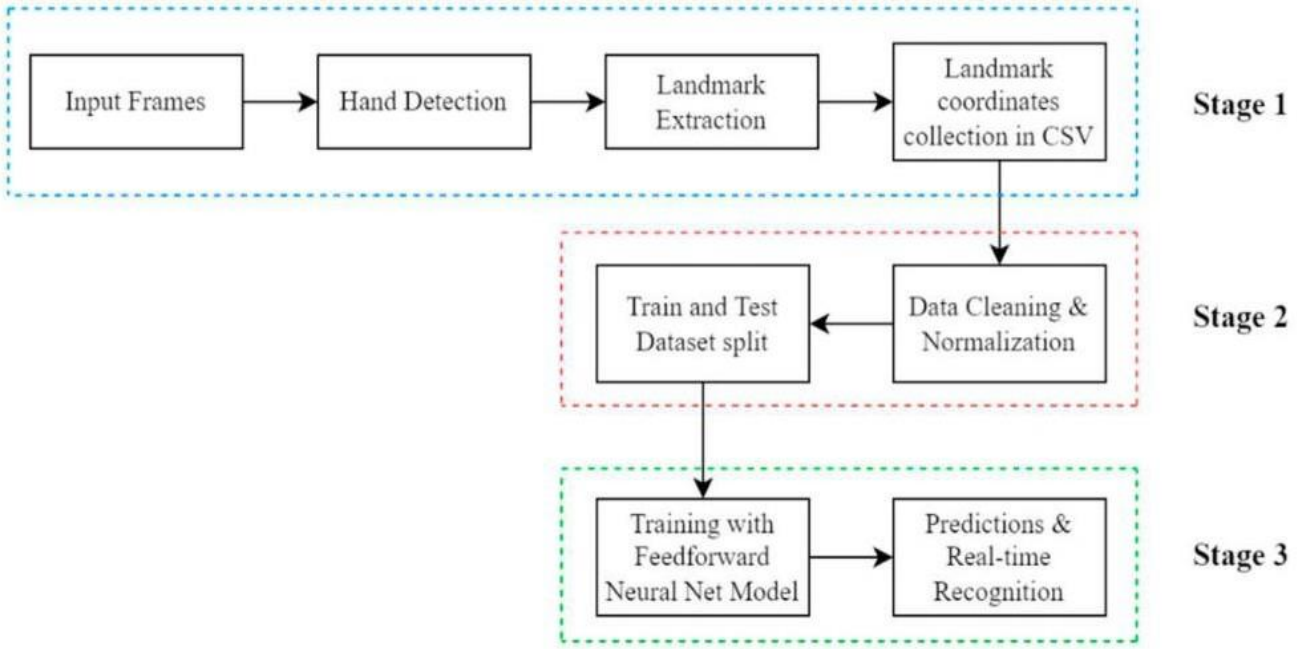
3. **Indian Sign Language Character Recognition**
   Authors: Prof.Amitabha Mukerjee

Every image is treated as a document. To define the words for the image, we use SIFT feature detectors to detect patches(or interest points) from the image and the SIFT feature descriptors return 128 dimensional vectors for each of the patches. Thus every image is a collection of 128 dimensional vectors.Now we convert these vector represented patches to codewords which produces a codebook(analogous to dictionary of words in text).Then we extracted the hog features of the images which returned us 32000 dimension vectors. Since we considered it too large, we used Gaussian Random Projection library in python to project the hog vectors into a 3000 dimensionalsubspace. On these vectors, we trained multiclass SVM with linear kernels which shot up the 4 fold CV accuracy to 51.43%.

THEORITICAL ANALYSIS

BLOCK DIAGRAMS




Input Frames
Hand Detection
Landmark Extraction
Landmark coordinates collection in CSV
Stage 1
Train and Test Dataset split
Data Cleaning & Normalization
Stage 2
Training with Feedforward Neural Net Model
Predictions & Real-time Recognition
Stage 3

EXPERIMENTAL INVESTIGATIONS

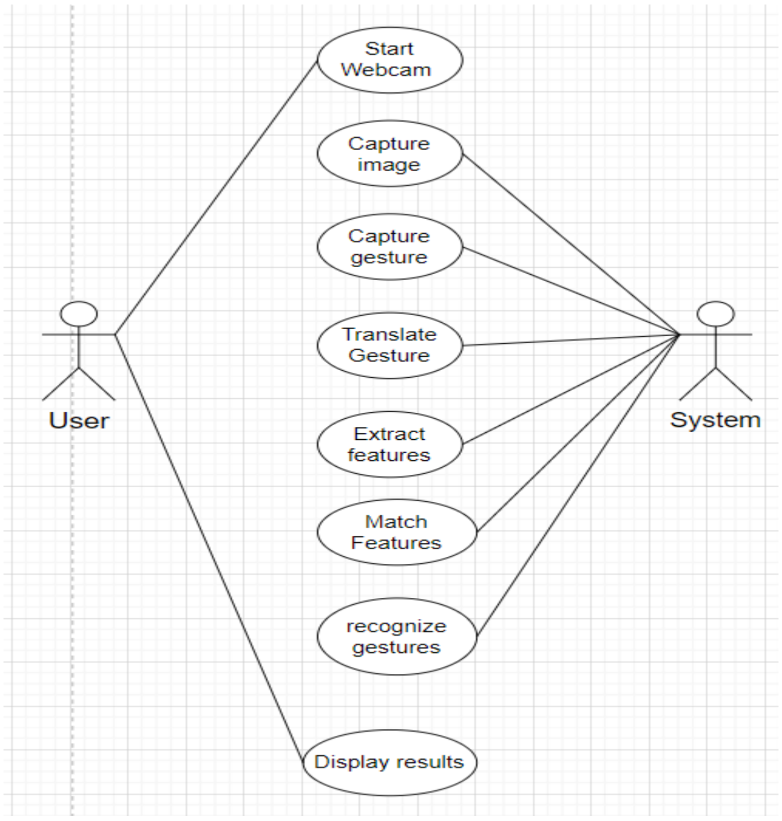
During the experimental investigations for a sign language detection project, several key analyses and investigations are typically conducted. Here are some common areas of focus:

Dataset Collection and Annotation: The first step involves gathering a diverse and representative dataset of sign language gestures. This dataset may consist of videos or images capturing different signs performed by individuals proficient in sign language. The dataset needs to be carefully annotated, specifying the corresponding sign for each sample.

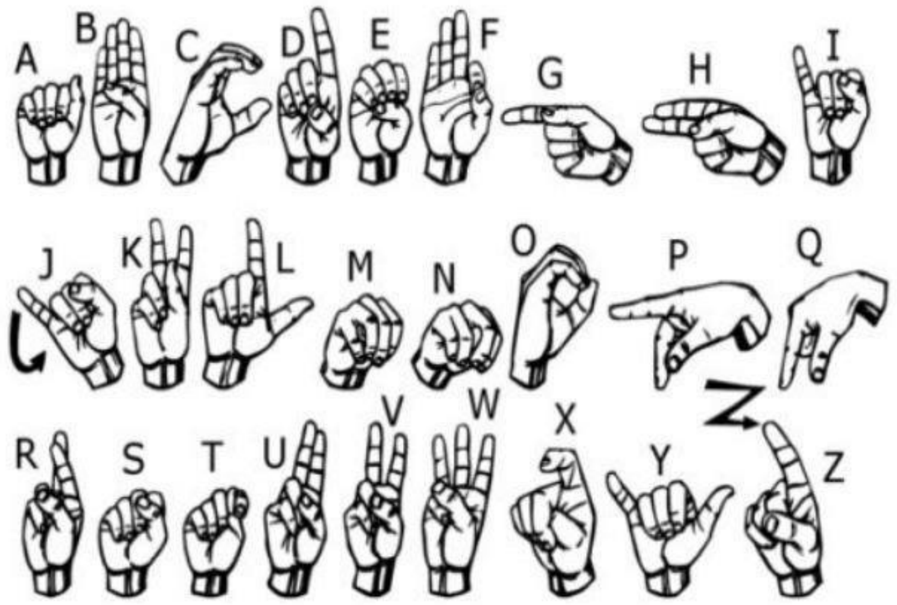
Machine Learning Algorithms: Different machine learning algorithms are trained and evaluated on the dataset. Common techniques include deep learning models such as Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), or more advanced architectures like Transformers. The model's ability to detect and classify sign language gestures accurately is assessed through metrics such as accuracy, precision, recall, and F1 score.

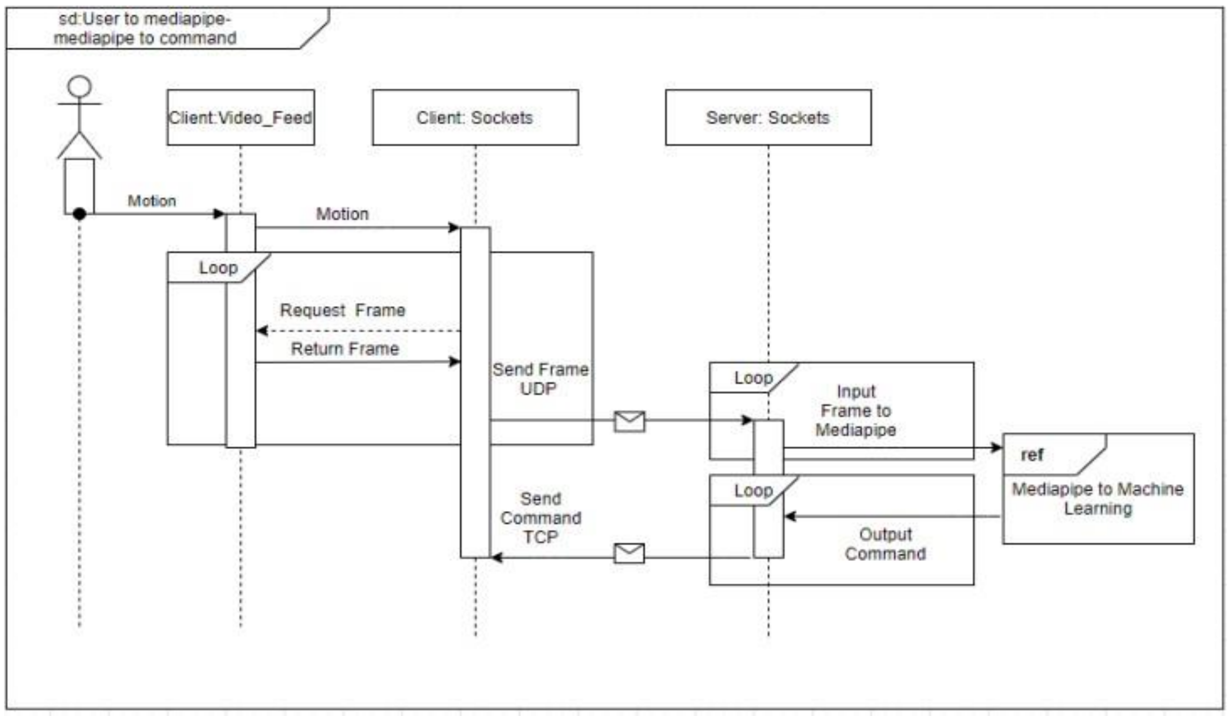
Performance Evaluation: The performance of the sign language detection system is thoroughly evaluated. This includes assessing how well the model generalizes to unseen data and its ability to handle variations in lighting conditions, occlusions, and different signers. Cross-validation techniques, such as k-fold cross-validation, may be employed to obtain reliable performance metrics.

FLOWCHART



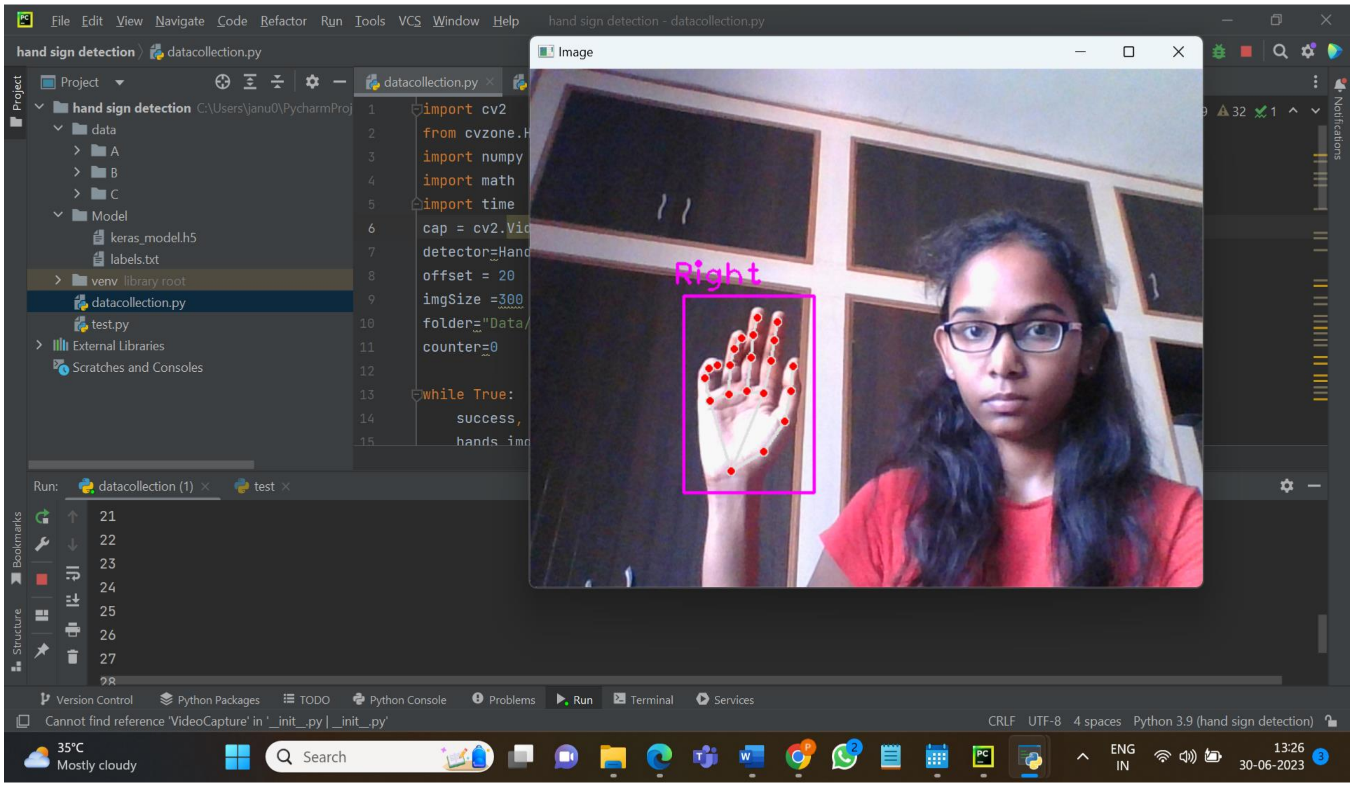
Sign Language





RESULT:

Data Collection

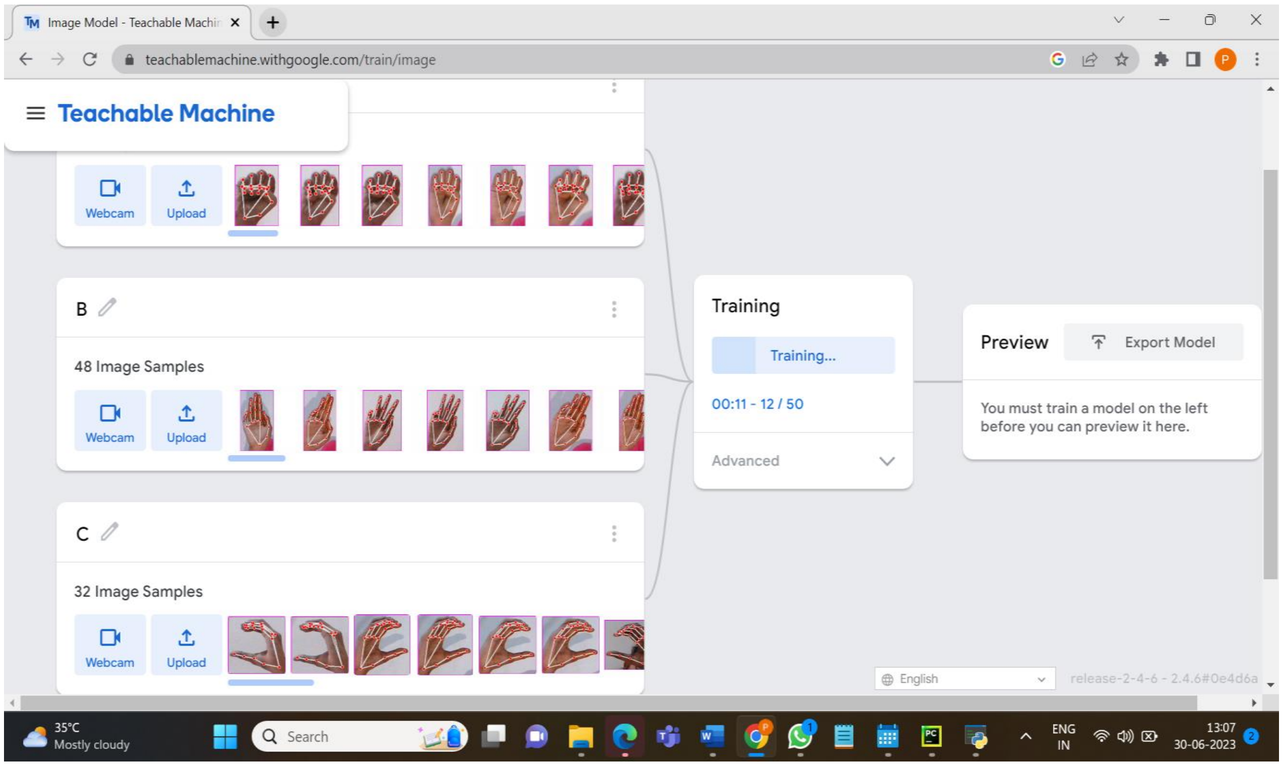


```
import cv2
from cvzone.H
import numpy
import math
import time
cap = cv2.Vid
detector=Hand
offset = 20
imgSize =300
folder="Data/
counter=0

while True:
    success,
    hands, img
```

```
21
22
23
24
25
26
27
28
```

Cannot find reference 'VideoCapture' in '__init__.py | __init__.py'

After this we have trained the images collected before using the Teachable Machine which converts the images into the Keras Model.



teachablemachine.withgoogle.com/train/image

B

48 Image Samples

C

32 Image Samples

Training

Training...

00:11 - 12 / 50

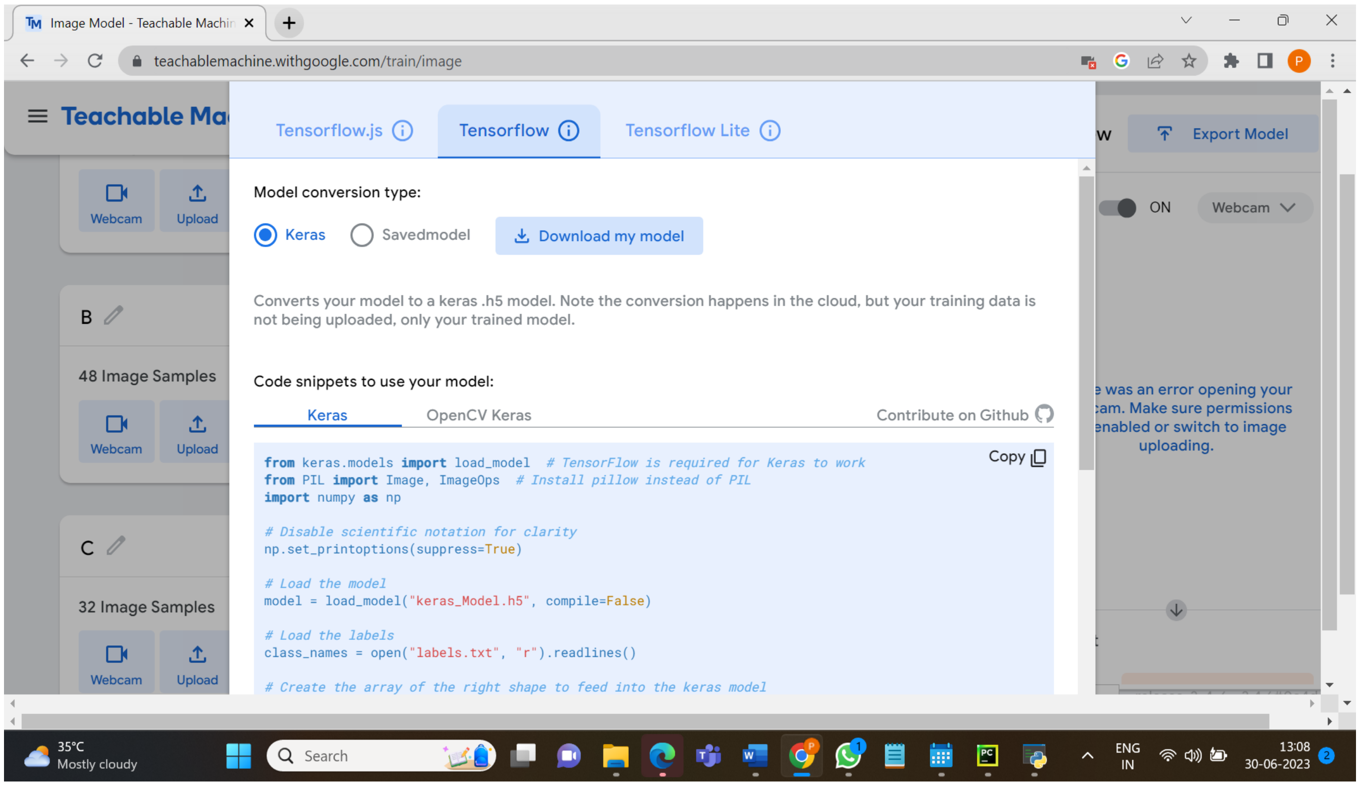
Advanced

Preview

Export Model

You must train a model on the left before you can preview it here.

Then the keras will be imported in the PyCharm and detect the signs incorporated before.



Model conversion type:

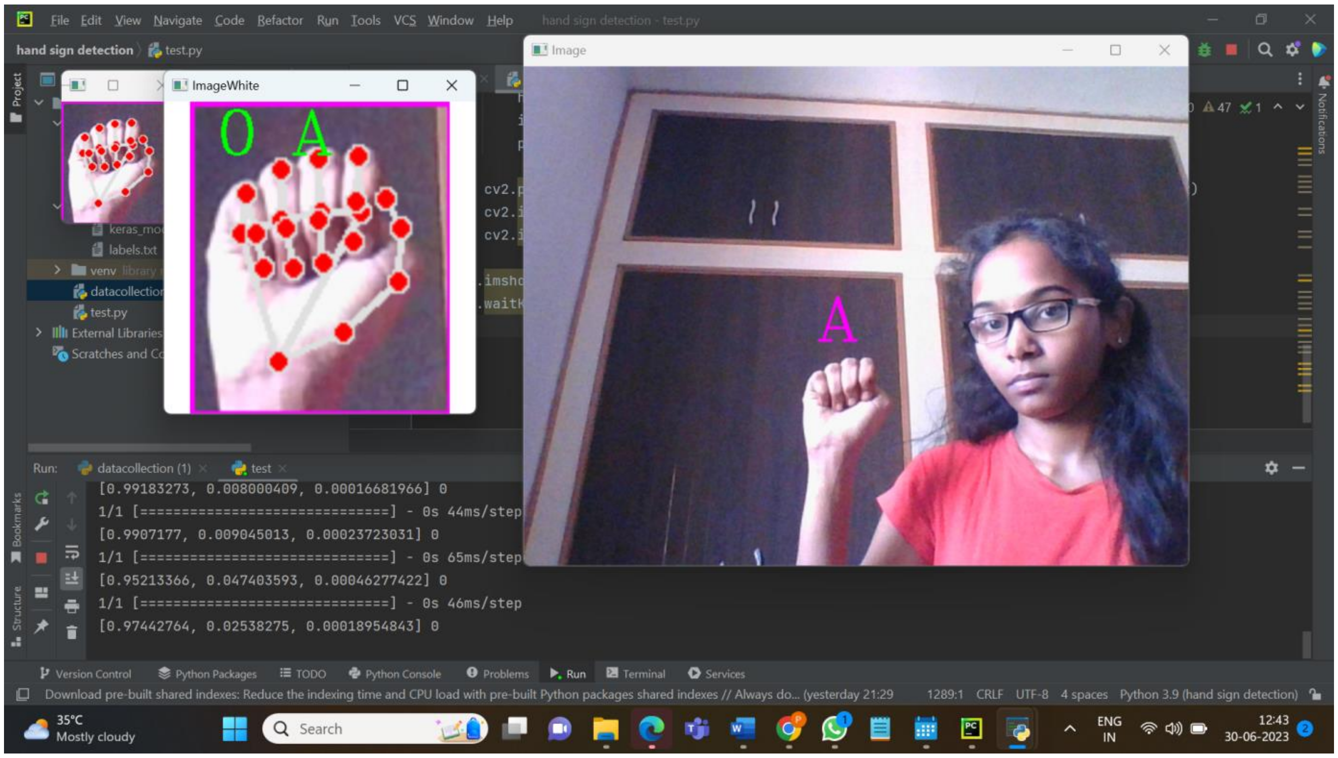
(● Keras) ○ Savedmodel

Converts your model to a keras .h5 model. Note the conversion happens in the cloud, but your training data is not being uploaded, only your trained model.

Code snippets to use your model:

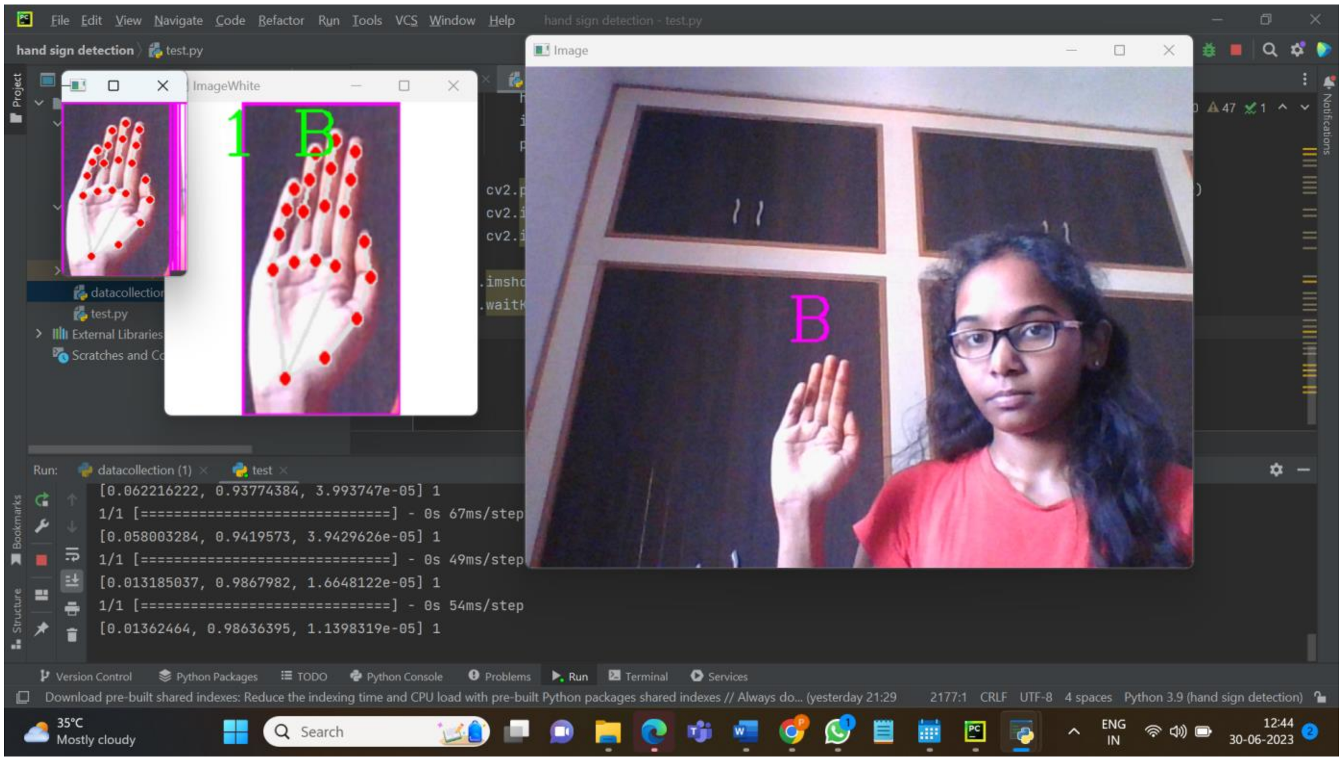
```
from keras.models import load_model  # TensorFlow is required for Keras to work
from PIL import Image, ImageOps  # Install pillow instead of PIL
import numpy as np

# Disable scientific notation for clarity
np.set_printoptions(suppress=True)

# Load the model
model = load_model("keras_Model.h5", compile=False)

# Load the labels
class_names = open("labels.txt", "r").readlines()

# Create the array of the right shape to feed into the keras model
```

Hand Sign Detection :

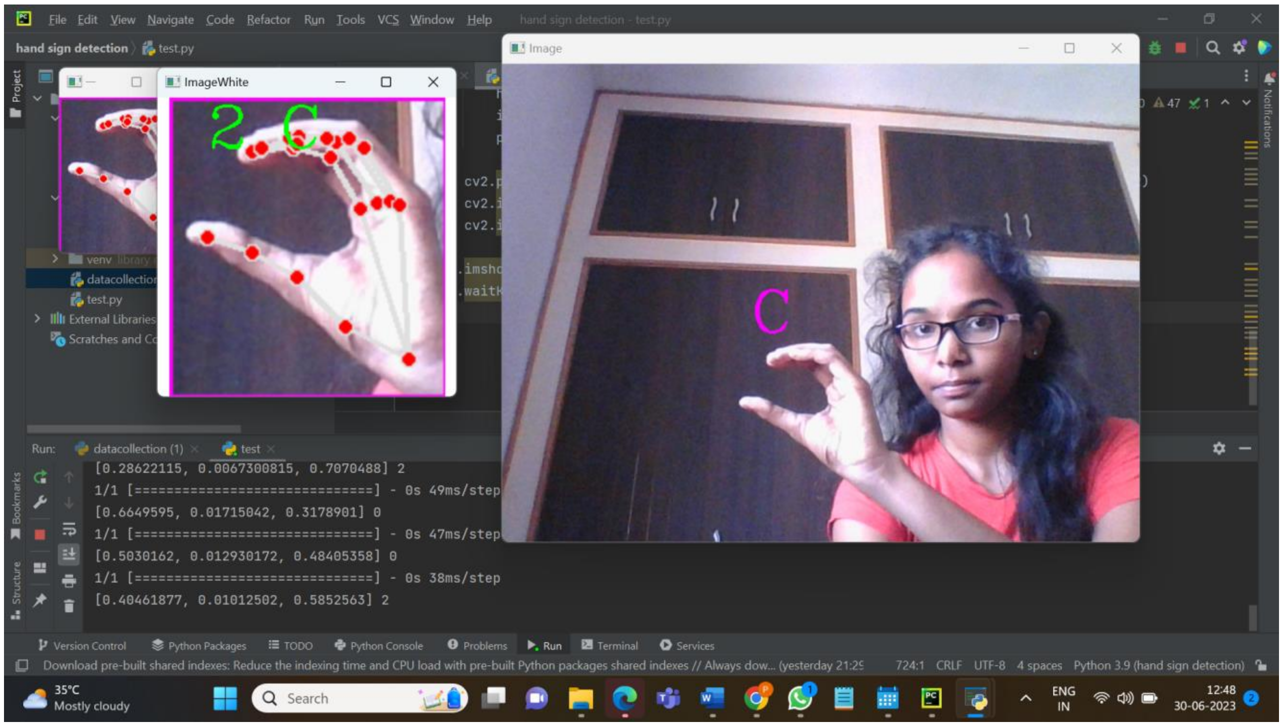


```
[0.99183273, 0.008000409, 0.00016681966] 0
1/1 [==============================] - 0s 44ms/step
[0.9907177, 0.009045013, 0.00023723031] 0
1/1 [==============================] - 0s 65ms/step
[0.95213366, 0.047403593, 0.00046277422] 0
1/1 [==============================] - 0s 46ms/step
[0.97442764, 0.02538275, 0.00018954843] 0
```

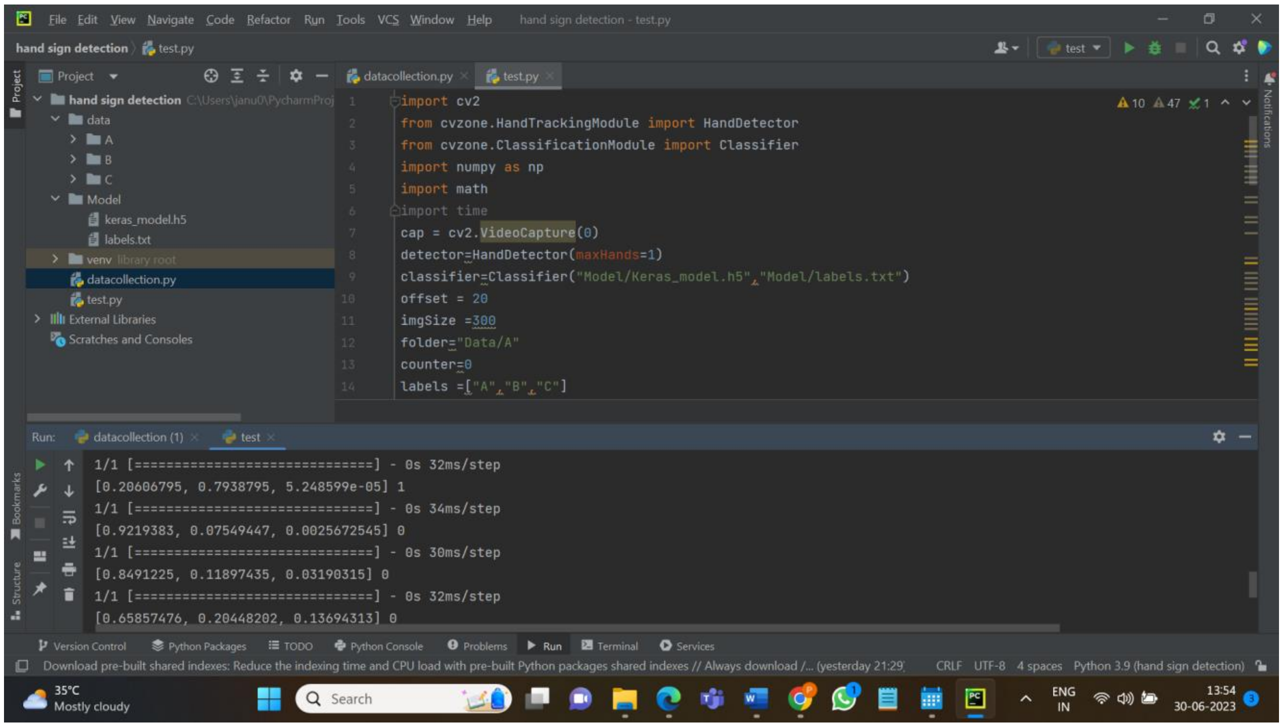
hand sign detection
test.py
ImageWhite
Image
A
labels.txt
datacollection
External Libraries
Scratches and Co
Run: datacollection (1) test
Version Control
Python Packages
TODO
Python Console
Problems
Run
Terminal
Services
Download pre-built shared indexes: Reduce the indexing time and CPU load with pre-built Python packages shared indexes // Always do... (yesterday 21:29
Python 3.9 (hand sign detection)
12:43 30-06-2023



```
[0.062216222, 0.93774384, 3.993747e-05] 1
1/1 [==============================] - 0s 67ms/step
[0.058003284, 0.9419573, 3.9429626e-05] 1
1/1 [==============================] - 0s 49ms/step
[0.013185037, 0.9867982, 1.6648122e-05] 1
1/1 [==============================] - 0s 54ms/step
[0.01362464, 0.98636395, 1.1398319e-05] 1
```



```
[0.28622115, 0.0067300815, 0.7070488] 2
1/1 [==============================] - 0s 49ms/step
[0.6649595, 0.01715042, 0.3178901] 0
1/1 [==============================] - 0s 47ms/step
[0.5030162, 0.012930172, 0.48405358] 0
1/1 [==============================] - 0s 38ms/step
[0.40461877, 0.01012502, 0.5852563] 2
```



```python
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector=HandDetector(maxHands=1)
classifier=Classifier("Model/Keras_model.h5","Model/labels.txt")
offset = 20
imgSize =300
folder="Data/A"
counter=0
labels =["A","B","C"]
```

```
1/1 [==============================] - 0s 32ms/step
[0.20606795, 0.7938795, 5.248599e-05] 1
1/1 [==============================] - 0s 34ms/step
[0.9219383, 0.07549447, 0.0025672545] 0
1/1 [==============================] - 0s 30ms/step
[0.8491225, 0.11897435, 0.03190315] 0
1/1 [==============================] - 0s 32ms/step
[0.65857476, 0.20448202, 0.13694313] 0
```

ADVANTAGES :

Accessibility: A sign language detection project can greatly enhance accessibility for the deaf and hard of hearing community. It can enable them to communicate with others more effectively and participate more fully in various aspects of society, including education, employment, and social interactions.

Communication Bridge: Sign language detection technology can act as a bridge between the deaf and hearing communities, facilitating smoother communication and reducing barriers. It allows sign language users to interact with non-sign language users more easily, promoting inclusivity and understanding.

Real-Time Communication: By detecting and translating sign language gestures in real time, the project can enable more immediate and fluid conversations between sign language users and others who do not understand sign language. This can be particularly beneficial in situations where a sign language interpreter may not be readily available.

Education and Learning: A sign language detection project can be integrated into educational settings to assist in teaching sign language to non-signers or provide additional support to sign language learners. It can enhance the learning experience and make it more interactive and engaging.

DISADVANTAGES:

Complexity of Sign Language: Sign languages are complex and diverse, with variations across regions and communities. Developing a sign language detection system that can accurately interpret and recognize all these variations can be challenging and require significant research and development.

Cost and Infrastructure: Developing and deploying sign language detection technology can be costly, requiring resources for research, development, and implementation. Additionally, it may require appropriate infrastructure, such as cameras or sensors, which may not be readily available or affordable in all settings.

Ethical Considerations: Privacy concerns and ethical considerations arise when deploying sign language detection systems. Proper measures should be taken to ensure user consent, data protection, and the avoidance of any discriminatory biases in the technology.

APPLICATIONS-

Communication Accessibility: Sign language detection can help bridge the communication gap between hearing individuals and deaf or hard-of-hearing individuals. By detecting and interpreting sign language

Education and Learning: Sign language detection can be used in educational settings to aid in the teaching and learning of sign language. It can provide feedback and corrections to learners, helping them improve their signing skills.

Human-Computer Interaction: Sign language detection can enhance human-computer interaction by allowing users to interact with computers, smartphones, or other devices using sign language gestures. This can be particularly beneficial for individuals with disabilities, enabling them to control devices, navigate interfaces, and access digital content using sign language.

Sign Language Research: Sign language detection can contribute to linguistic research and analysis of sign languages. By accurately tracking and interpreting sign language gestures, researchers can study the structure, grammar, and syntax of sign languages, leading to a better understanding of these languages and their cultural significance.

FUTURE SCOPE:

Enhanced Accuracy: Improving the accuracy of sign language detection systems is a key focus for the future. Advancements in computer vision and machine learning techniques can lead to more precise recognition of sign language gestures, reducing errors and increasing overall system performance.

Real-Time Translation: Real-time sign language translation is an area that can benefit from further development. As technology progresses, sign language detection systems can be integrated with machine translation algorithms to provide instant translation of sign language gestures into spoken or written language, enabling seamless communication in various settings.

Multi-modal Integration: The integration of multiple modalities, such as combining sign language detection with speech recognition or facial expression analysis, can enhance the overall understanding and interpretation of sign language. This can lead to more accurate and context-aware translations, considering the nuances conveyed through facial expressions and additional spoken language.

CONCLUSION:

With an average accuracy of 99% in most of the sign language dataset using MediaPipe's technology and machine learning, our proposed methodology show that MediaPipe can be efficiently used as a tool to detect complex hand gesture precisely. Although, sign language modelling using image processing techniques has evolved over the past few years but methods are complex with a requirement of high computational power. Time consumption to train a model is also high. From that perspective, this work provides new insights into this problem. It is achieved by constructing a custom data set, making the system invariant to rotation and solving the background dependency problem.

BIBILOGRAPHY:

[1] Bantupalli Kshitij, Xie Ying. American sign language recognition using machine learning and computer vision. Master of Science in Computer Science Theses 2019; 21.

[2] Shivashankara S, Srinath S. A comparative study of various techniques and outcomes of recognizing American sign language: a review. In: International Journal of Scienti・c Research Engineering & Technology (IJSRET); 2017. ISSN 2278 – 0882. 6(9).

[3] Viswanathan Daleesha M, Idicula Sumam Mary. Recent developments in Indian sign language recognition: an analysis. Int J Comput Sci Inf Technol 2015;6(1): 289–93.

[4] Nair Anuja V, Bindu V. A review on Indian sign language recognition. Int J Comput Appl 2013;73(22). [6] Athira K, Sruthi CJ, Lijiya A. A signer independent sign language recognition with Co-articulation elimination from live videos: an Indian scenario. J King Saud Univ Comput Inf Sci 2022;34(3):771–8.

APPENDIX

SOURCE CODE:

Datacollection.py

```python
import cv2
from cvzone.HandTrackingModule import HandDetector
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector=HandDetector(maxHands=1)
offset = 20
imgSize =300
folder="Data/A"
counter=0

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]
        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
        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)
    if key ==ord("s"):
        counter +=1
        cv2.imwrite(f'{folder}/Image_{time.time()}.jpg',imgWhite)
        print(counter)
```

test.py

```python
import cv2
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import numpy as np
import math
import time
cap = cv2.VideoCapture(0)
detector=HandDetector(maxHands=1)
classifier=Classifier("Model/Keras_model.h5","Model/labels.txt")
offset = 20
imgSize =300
folder="Data/A"
counter=0
labels =["A","B","C"]
while True:
    success, img = cap.read()
    imgOutput= img.copy()
    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]
        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)
            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(img)

        cv2.putText(imgOutput,labels[index],(x,y-
20),cv2.FONT_HERSHEY_COMPLEX,2,(255,0,255),2)
        cv2.imshow("ImageCrop",imgCrop)
        cv2.imshow("ImageWhite",imgWhite)

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

THANK YOU