A PROJECT REPORT ON

HAND SIGN LANGUAGE DETECTION AND VIDEO GENERATION

For

TECHVARIABLE PRIVATE LIMITED, CHANDMARI, GUWAHATI

AND

SUBMITTED IN PARTIAL FULFILLMENT OF THE REQUIREMENT FOR THE DEGREE OF MASTER OF COMPUTER APPLICATION (MCA)

Submitted By:

SUMU AHMED

Roll No: 220320043028

Reg No: 004003222

INTERNAL GUIDE -

DR. MUKTA RANJAN SINGHA

Associate Professor & HOD

GIMT

EXTERNAL GUIDE -

ABHISHEK CHATTERJEE

Senior Software Engineer

TechVariable

DEPARTMENT OF COMPUTER APPLICATION

GIRIJANANDA CHOWDHURY INSTITUTE OF MANAGEMENT AND TECHNOLOGY

ASSAM SCIENCE AND TECHNOLOGY UNIVERSITY

JUN– 2024

Declaration by the Candidate

I Sumu Ahmed, Roll No 220320043028 and students of the Department of Computer Application, Girijananda Chowdhury Institute of Management and Technology hereby declares that we have compiled this progress report reflecting all our works during the semester long full time project as part of our MCA curriculum.

We declare that we have included the descriptions of our project work, and nothing has been copied/replicated from other's work. The facts, figures, analysis, results, claims etc. depicted in my reports are all related to our full time project work.

We also declare that the same report or any substantial portion of this report has not been submitted anywhere else as part of any requirements for any degree/diploma etc.

Sumu Ahmed
Signature
Date

ACKNOWLEDGEMENT

We thank Mr. Arindom Ain, Assistant Professor, Mr. Rajashri Paul, Assistant Professor & System Analyst, Mr. Jeetumoni Barman, Assistant Professor, Dr. Arnab Kumar Das, Assistant Professor, Mr. Pranjal Das, Assistant Professor, Department of Computer Application, GIMT, for their constant support and encouragement given throughout the development of the project. We take this opportunity to specially extend our gratitude to our Project Guide, Dr. Mukta Ranjan Singha, Associate Professor & HOD, Department of Computer Application, GIMT. Last but not the least, our sincere thanks to our parents, family and friends for their continuous support, inspiration, and encouragement without which this project would not have been successful.

Signature

Sumu Ahmed
MCA 4th Semester
Roll Number: 220320043028

Department Of Computer Applications
GIMT, Guwahati

Certificate

This is to certify that Sumu Ahmed have carried out the project work "**HAND SIGN LANGUAGE DETECTION AND VIDEO GENERATION**" under my supervision and have compiled this project report reflecting the candidates work in the semester long project. The candidates did this project full time during the whole semester under my supervision, and the analysis, results, claims etc. are all related to their studies and works during the semester.

I recommend submission of this project report as a part for partial fulfillment of the requirements for the degree of Master of Computer Application of Assam Science and Technology University.

Internal/External Guide

GIRIJANANDA CHOWDHURY INSTITUTE OF MANAGEMENT AND TECHNOLOGY

DEPARTMENT OF COMPUTER APPLICATION

Certificate

This is to certify that **Sumu Ahmed,** Roll No: **220320043028** have successfully completed the System Development Project Work on "**Hand Sign Language Detection and Video Generation"** at **Girijananda Chowdhury Institute of Management and Technology, Guwahati**. This project work is a bonafide work for the partial fulfilment of the requirement of the third semester of **Master of Computer Application** Programme in the Department of Computer Application, **Girijananda Chowdhury Institute of Management and Technology**, Guwahati, affiliated to **Assam Science And Technology University, Guwahati** and approved by **AICTE**, MHRD, Govt. of India

The work done by them is an academic work and cannot be presented for any other purpose.

I wish them all success in life.

External Examiner

Head
Department of Computer Application
GIMT, Guwahati

INTERNSHIP COMPLETION CERTIFICATE

TABLE OF CONTENTS

INTRODUCTION

Although sign languages have emerged naturally in deaf communities alongside or among spoken languages, they are unrelated to spoken languages and have different grammatical structures at their core. Sign languages are fully-fledged natural languages with their own syntax, morphology, and lexicons. One might expect digital technologies to play a huge role in human daily routines, and the whole world may soon interact via machines, using gestures or speech recognition within a few decades. If we are in a position to predict such a future, we ought to think about the physically challenged and do something for them.

Sign language is the natural language of the deaf and aphonic people, serving as the primary method of communication for the deaf community. There are numerous different sign languages in the world, each with unique characteristics and regional variations. Some of the most well-known sign languages include American Sign Language (ASL), Indian Sign Language (ISL), German Sign Language (GSL), Chinese Sign Language (CSL), Australian Sign Language (Auslan), Arabic Sign Language (ArSL), and many more. Each of these languages has evolved to meet the communication needs of its users within different cultural and linguistic contexts.

In this project, the focus lies on American Sign Language (ASL), a comprehensive language utilizing hand movements. ASL, distinct from English, possesses its own grammatical structure. The project's objective is to develop a system leveraging machine learning to accurately interpret ASL gestures. By utilizing techniques such as Convolutional Neural Networks (CNNs) and computer vision, the system aims to enhance communication for the deaf and hard of hearing. Through real-time gesture capture via webcam and CNN-based prediction, the system can bridge communication barriers. Future applications may include real-time translation and enhanced accessibility features.

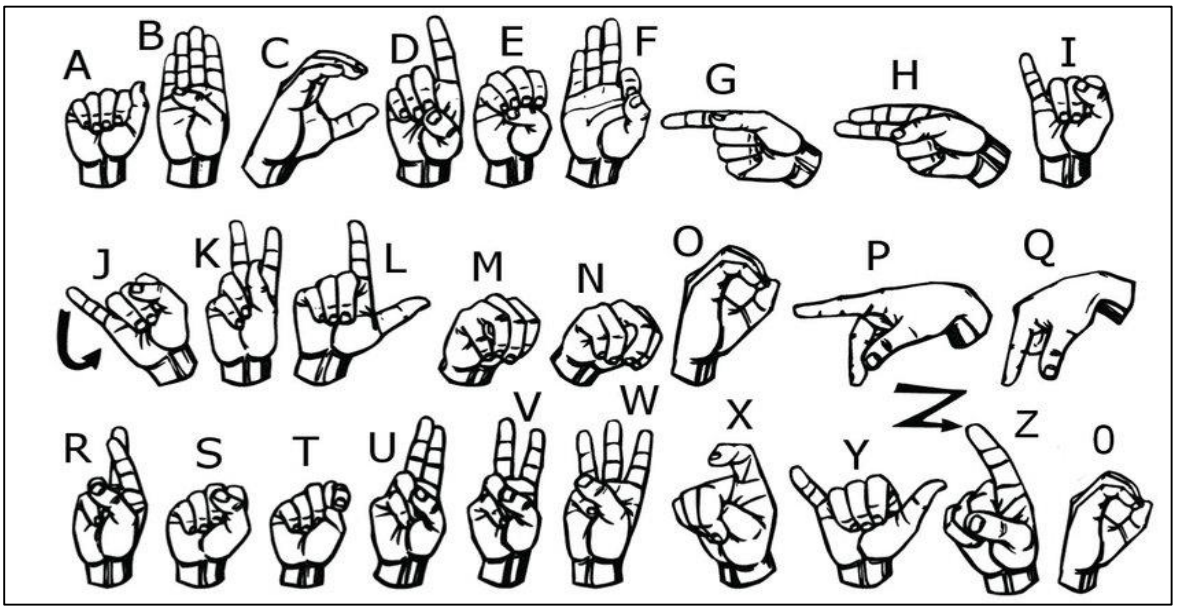


Fig: ASL Hand Signs

OBJECTIVES

The main objectives of this project are to contribute to the field of automatic sign language recognition and translation to text or speech. In our project, we focus on static sign language hand gestures. This work focused on recognizing the hand gestures which includes 26 English alphabets (A-Z) using Deep Neural Networks (DNN). We created a convolution neural networks classifier that can classify the hand gestures into English alphabets. We have trained the neural network under different configurations and architectures like Google Colab, CNN and our own architecture. We used the horizontal voting ensemble technique to achieve the maximum accuracy of the model. We have also created a web application using Streamlit Frameworks to test our results from a live camera.

SYSTEM FEASIBILITY

Economic Feasibility: The economic feasibility involves conducting a cost analysis to determine the budget needed for hardware, such as webcams and computers, and software, including any necessary licensing for libraries like TensorFlow and OpenCV. Additionally, a benefit analysis is essential to evaluate the potential advantages of the system, such as improved communication for hearing-impaired individuals and potential cost savings compared to traditional communication methods.

Technical Feasibility: Technical feasibility focuses on the availability and compatibility of the required technology. This includes ensuring that tools like TensorFlow, OpenCV, and Streamlit are accessible and can be integrated with existing systems. It is also important to assess the technical skills of the development team to determine if additional training or hiring is needed to implement and maintain the system effectively.

Operational Feasibility: Operational feasibility examines the acceptance and integration of the system into daily operations. It is important to gauge the willingness of potential users, such as hearing-impaired individuals, to adopt the new system. Additionally, the system should be designed to integrate smoothly with existing workflows and operations without causing significant disruptions

TECHNOLOGY AND TOOLS

HARDWARE REQUIREMENT

- **Processor**:
  - Minimum: Quad-core CPU or higher
- **Memory (RAM)**:
  - Minimum: 8 GB
  - Recommended: 16 GB or higher
- **Graphics Processing Unit (GPU)**:
  - Optional for smaller models and datasets
  - Recommended for training large models or working with large datasets:
    - NVIDIA GPU with CUDA support (e.g., GTX 1060 or higher)
- **Storage**:
  - Minimum: 20 GB free disk space
  - Recommended: SSD with 100 GB or more free disk space for faster data processing
- **Camera**:
  - A high-resolution webcam or an external camera for capturing hand gestures.

SOFTWARE REQUIREMENT

- **Operating System**:
  - Windows 10 or later
  - macOS 10.14 or later
  - Linux (Ubuntu 18.04 or later recommended)
- **Programming Language**:
  - Python 3.6 or later
- **Libraries and Frameworks**:
  - TensorFlow
  - Keras
  - OpenCV
  - Media Pipe
  - NumPy
  - Matplotlib (for visualizations)
  - Streamlit
- **Development Environment**:
  - PyCharm IDE
  - Google Colab

Libraries and Frameworks that I used:

TensorFlow

TensorFlow is the foundational framework used for developing the machine learning model in this project. It provides comprehensive support for constructing and training deep learning models, especially Convolutional Neural Networks (CNNs). TensorFlow offers a vast library of tools and pre-built functions that simplify the creation of complex neural network architectures. It supports efficient training and deployment of models on various hardware platforms, ensuring scalability and performance. TensorFlow's ability to handle large-scale machine learning tasks makes it an essential framework for developing robust and accurate models.

Keras

Keras is a high-level neural networks API, written in Python and capable of running on top of TensorFlow. It provides a user-friendly interface for constructing and compiling neural networks. Keras simplifies the process of assembling various layers, such as convolutional, pooling, and dropout layers, enabling rapid prototyping and experimentation. Its intuitive API allows for quick and easy model development, reducing the complexity of coding deep learning models. Keras is ideal for both beginners and experts due to its simplicity and flexibility.

OpenCV

OpenCV (Open Source Computer Vision Library) is employed for a wide range of image processing tasks. In this project, it is used to capture and preprocess hand sign images, ensuring they are in the correct format and size for input into the neural network. OpenCV provides efficient functions for various image manipulation tasks, such as resizing images, applying filters, and converting color spaces. These preprocessing steps are crucial for preparing the dataset and ensuring that the images fed into the neural network are standardized and optimized for learning.

MediaPipe

MediaPipe is a versatile framework developed by Google for building multimodal machine learning pipelines. In this project, MediaPipe is utilized for real-time hand gesture detection and tracking. Its hand-tracking module provides accurate and efficient detection of hand landmarks, which are then used to generate input images for the CNN. By integrating MediaPipe, the system can capture dynamic hand gestures in real-time, enhancing the overall robustness and accuracy of the hand sign detection system. MediaPipe simplifies the implementation of real-time hand gesture recognition by providing pre-trained models and easy-to-use APIs.

NumPy

NumPy is a fundamental library for numerical computing in Python, extensively used for handling and manipulating arrays and matrices of image data. NumPy's powerful array operations enable efficient computation and transformation of image data, facilitating the preprocessing and preparation of data for the neural network. Its capabilities in managing large datasets and performing complex mathematical operations make it an indispensable tool in the data processing pipeline of this project.

Matplotlib

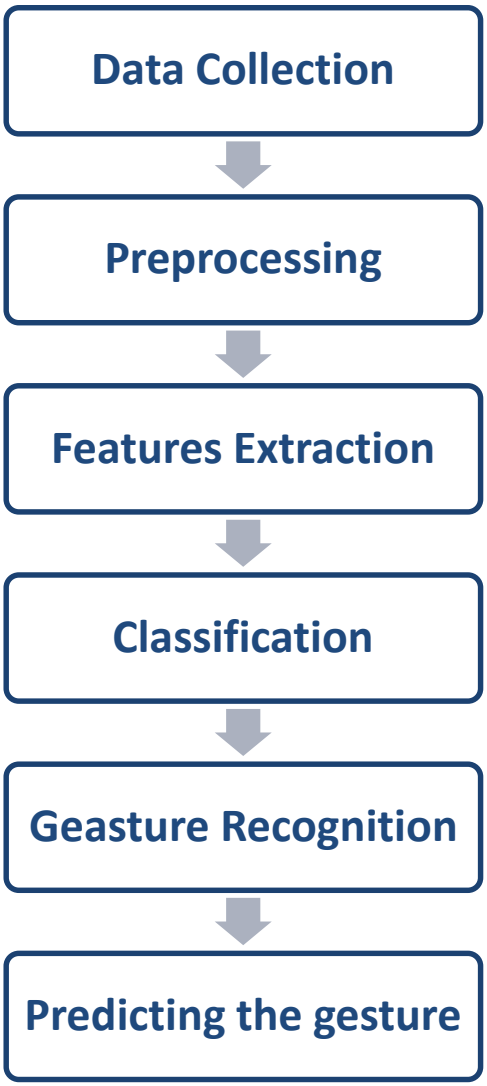
Matplotlib is a comprehensive library for creating static, animated, and interactive visualizations in Python. In this project, it is used for visualizing the training process and the performance of the model. Matplotlib helps in plotting graphs and charts that illustrate the accuracy and loss metrics over the training epochs. These visualizations provide valuable insights into the model's learning curve, highlighting potential issues such as overfitting or underfitting, and guiding further optimization of the model. Effective data visualization is crucial for understanding the behavior of the model and making informed decisions during the training process.

Streamlit

Streamlit is an open-source framework for creating interactive web applications. In this project, Streamlit is used to develop a user-friendly interface for demonstrating the hand sign language detection model. It allows users to upload images or use their webcam to perform real-time hand sign recognition. Streamlit's straightforward API makes it easy to deploy the model and create an engaging application for end-users to interact with the hand sign detection system. By providing a seamless way to share and demonstrate machine learning models, Streamlit enhances the accessibility and usability of the project.

METHODOLOGY

Any machine learning based application can be summed up to have at least three phases - data collection and preprocessing phase, training phase and visualization. Our program also follows these steps in order. At first, the data is collected and a base dataset is prepared. This dataset is then divided into training data and testing data which in our case is a multi-label classification data as we have to predict 26 gestures. To generate our dataset, we've collected hand key points from images for each gesture using the laptop's web camera. Features are then selected and extracted from the training data. The next step is to decide which machine learning models to use. Since ours is a multi-class classification problem, the models used different types of architecture. After that, these models are then trained on the training set. Then they are made to make predictions on the test set based on which 6 their performance is evaluated and changes are made to the parameters so as to squeeze out the best results from these models.

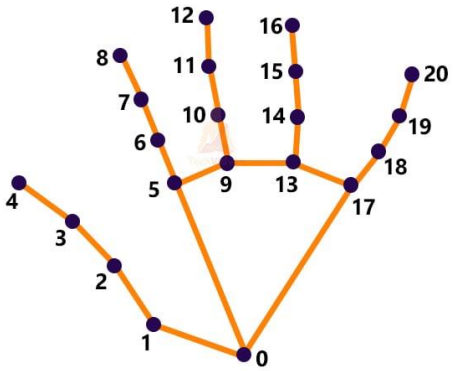


SYSTEM DEVELOPMENT

Dataset collection

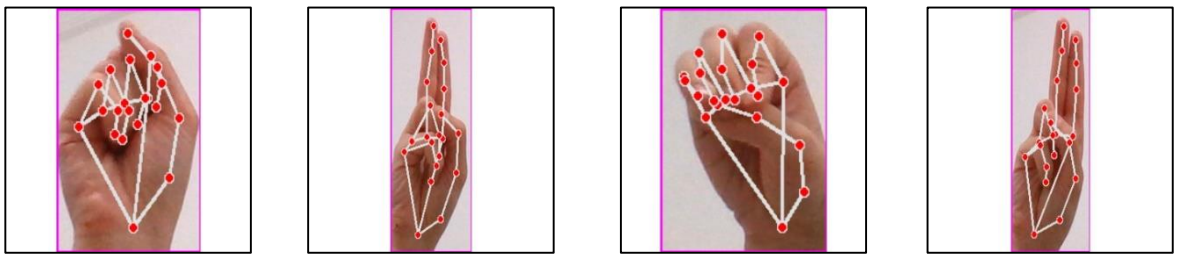
The first step in any machine learning problem is to gather the data. The data can either be taken from some open source datasets from websites such as Kaggle or you can prepare your own dataset. In our case, we created our own dataset from scratch. For the data gathering process, we took x- and y-coordinates of 21 hand keypoints using the MediaPipe and OpenCV libraries. For each gesture, the following x and y keypoints were collected:

- Wrist
- Thumb
- Index finger
- Middle finger
- Ring finger
- Pinky finger



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

For our project, we worked with a dataset comprising a total of 3000 images, each with a pixel size of 300x300.



Data Pre-processing

An image is nothing more than a 2-dimensional array of numbers or pixels which are ranging from 0 to 255.Typically, 0 means black, and 255 means white. Image is defined by mathematical function f(x, y) where 'x' represents horizontal and 'y' represents vertical in a coordinate plane. The value of f(x, y) at any point is giving the pixel value at that point of an image.

Image Pre-processing is the use of algorithms to perform operations on images. It is important to Pre-process the images before sending the images for model training. So in here we used "ImageDataGenerator" from TensorFlow to rescale the pixel value to the range [0,1] and split dataset into two categories training and validation and all the image size is reduced to 150x150 pixels from 300x300 pixels. If not, the model cannot be trained.

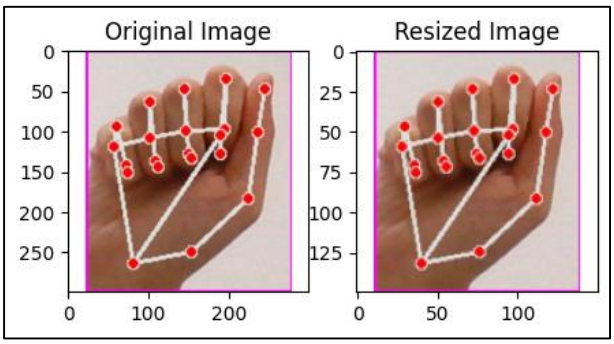
```
data_path = '/content/drive/MyDrive/mediadata/Data'
img_size = 150 #image size reduce to 150

# Data generator
datagen = ImageDataGenerator(rescale=1./255, validation_split=0.2)

# Training and validation generators
train_generator = datagen.flow_from_directory(
    data_path,
    target_size=(img_size, img_size),
    batch_size=32,
    class_mode='categorical',
    subset='training'
)

validation_generator = datagen.flow_from_directory(
    data_path,
    target_size=(img_size, img_size),
    batch_size=32,
    class_mode='categorical',
    subset='validation'
)
```

After resized the image to 150x150 pixels.



Designing the Convolution Neural Network (CNN) architecture

Computer Vision is a field of Artificial Intelligence that focuses on problems related to images and videos. CNN combined with Computer vision is capable of performing complex problems.

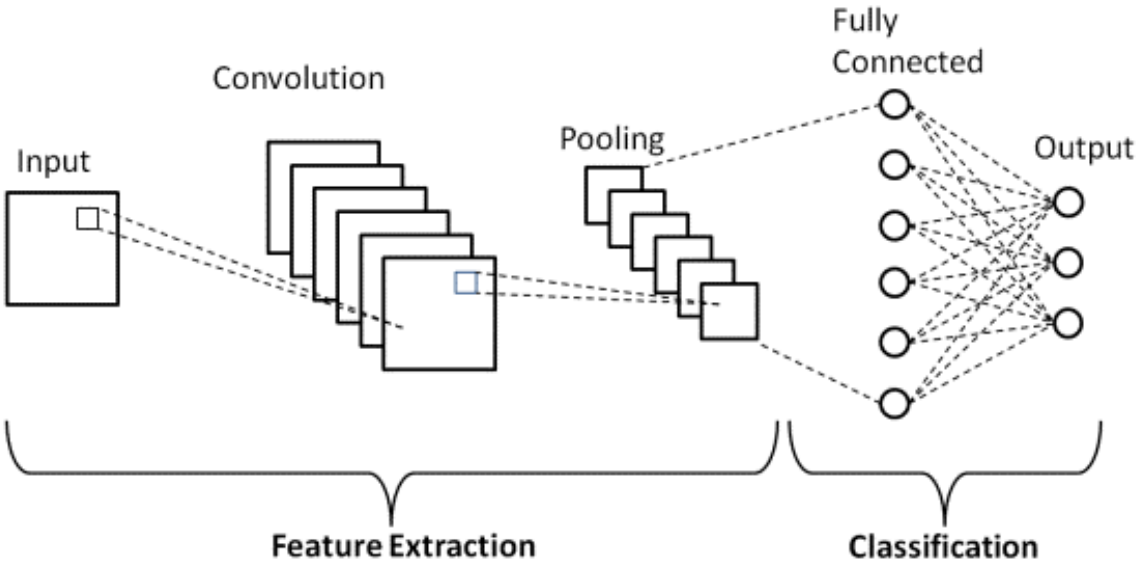


Fig: Working of CNN

The Convolution Neural Networks has two main phases namely feature extraction and Classification. A series of convolution and pooling operations are performed to extract the features of the image. The size of the output matrix decreases as we keep on applying the filters.

Size of new matrix = (Size of old matrix — filter size) +1

A fully connected layer in the convolution neural networks will serve as a classifier.

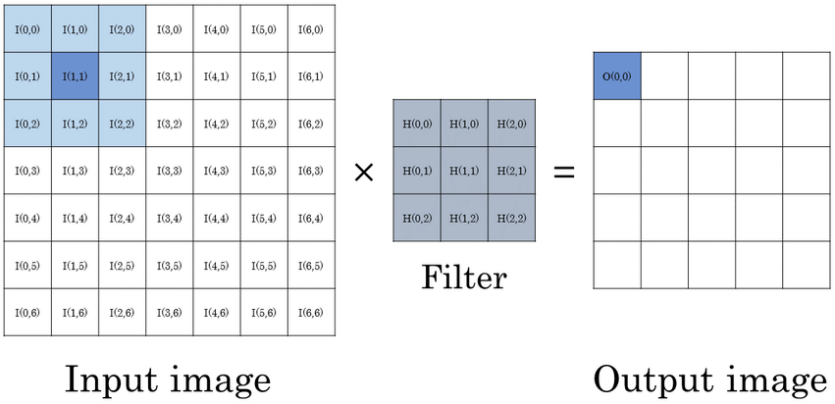
In the last layer, the probability of the class will be predicted.

The main steps involved in convolution neural networks are:

1) **Convolution**
2) **Pooling**
3) **Flatten**
4) **Fully Connected**

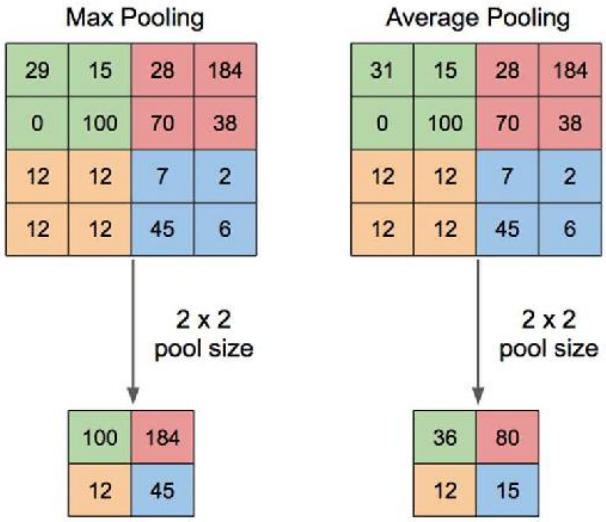
Input Layer:

The model begins with an initial convolutional layer that uses 32 filters of size 3x3. This layer processes the input images, which have dimensions of (img_size, img_size, 3), where 'img_size' represents the width and height of the images and 3 denotes the RGB color channels. The activation function used is ReLU (Rectified Linear Unit), which helps in introducing non-linearity into the model. This layer is crucial for detecting basic features such as edges and textures in the images.



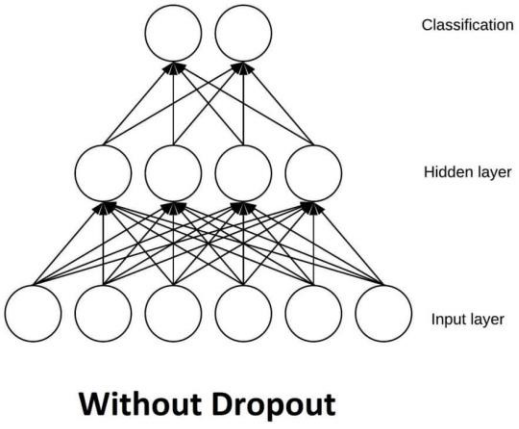
First Convolution and Pooling Block:

Following the input layer, the model includes a max-pooling layer with a pool size of 2x2. Max-pooling is a down-sampling operation that reduces the spatial dimensions of the feature maps, thereby reducing the computational load and helping the model to focus on the most significant features. This block allows the model to gain translational invariance, making it more robust to shifts in the input images.

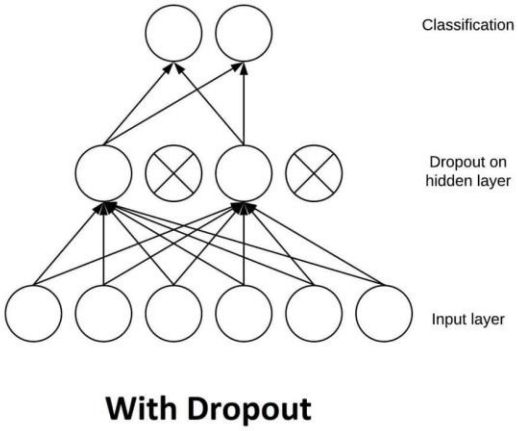


Second Convolution and Dropout Block:

The second convolutional layer has 64 filters of size 3x3, again with ReLU activation. This layer builds on the features detected by the first layer, allowing the model to learn more complex patterns. To prevent overfitting, a dropout layer with a dropout rate of 0.4 is added. Dropout is a regularization technique that randomly sets 40% of the input units to zero during training, which helps the model to generalize better by not relying too heavily on any single neuron. This block is followed by another max-pooling layer with a pool size of 2x2, further reducing the spatial dimensions.



Without Dropout



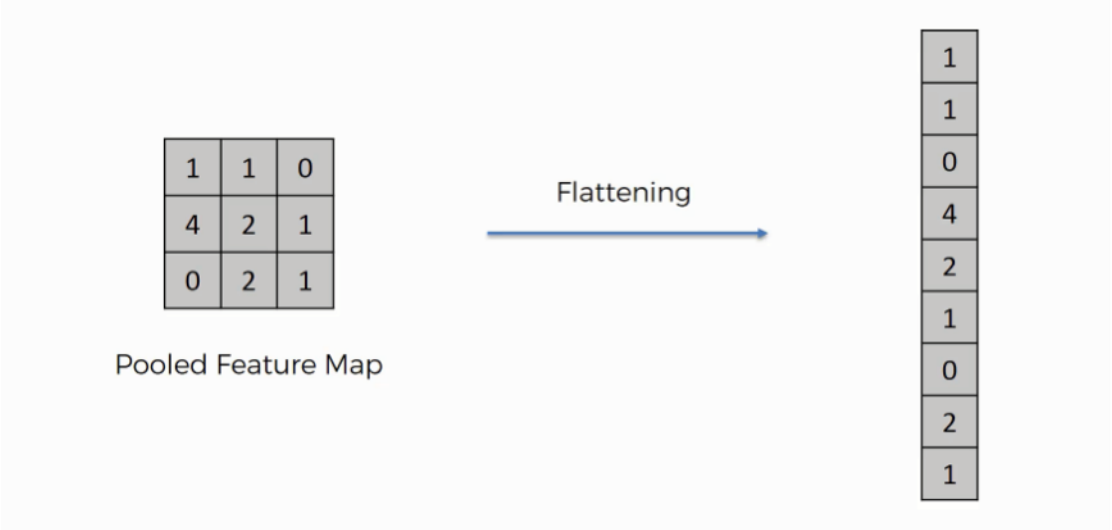
With Dropout

Third Convolution and Pooling Block:

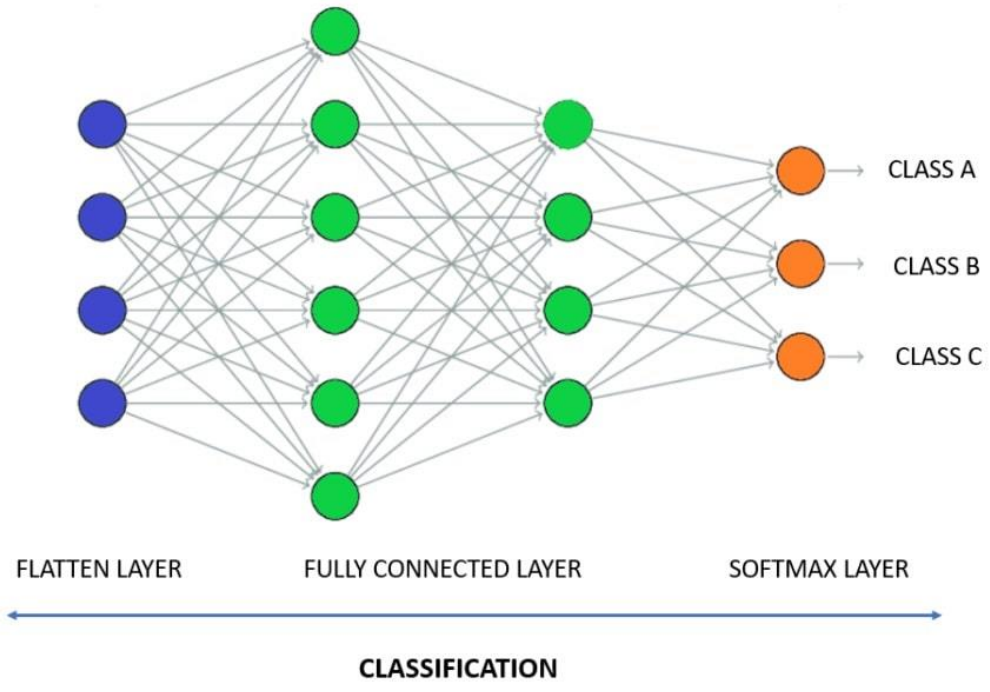
The third convolutional layer uses 128 filters of size 3x3, with ReLU activation. This layer captures even more detailed and abstract features from the images. Following this, another max-pooling layer with a pool size of 2x2 is applied. This series of convolutional and pooling layers helps in progressively reducing the spatial dimensions while increasing the depth of the feature maps, thereby consolidating the feature extraction process.

Flattening and Fully Connected Layers:

After the convolutional and pooling layers, the feature maps are flattened into a one- The obtained resultant matrix will be in muti-dimension. Flattening is converting the data into a 1-dimensional array for inputting the layer to the next layer. We flatten the convolution layers to create a single feature vector.



After flatten layer, this transformation prepares the data for the **Dense (fully connected**) layers, which operate on one-dimensional vectors. The first dense layer has 128 neurons with ReLU activation, allowing the model to learn high-level representations of the data. This layer is essential for combining the features detected by the convolutional layers into meaningful patterns that can be used for classification.

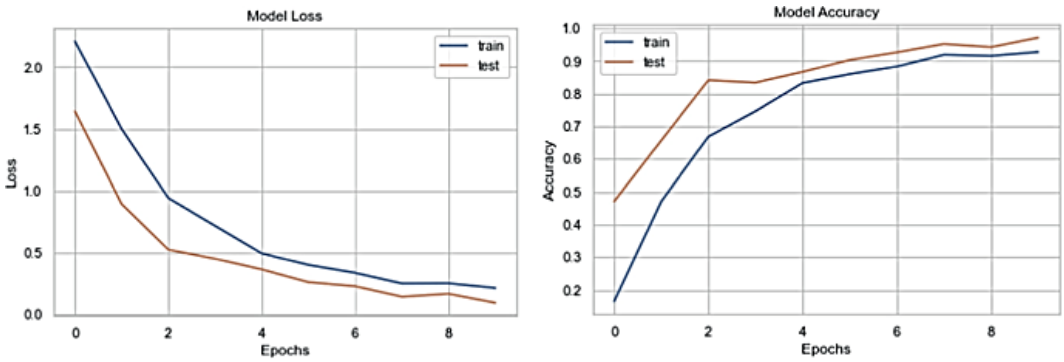


Model Compilation and Summary:

The model is compiled using the **Adam optimizer**, which is well-suited for handling sparse gradients and maintaining efficient training. The loss function used is **categorical cross-entropy,** ideal for multi-class classification problems. During training, the model's performance is monitored using accuracy metrics, ensuring that it learns to correctly identify hand signs from the training data.

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

Model Accuracy:



Model Performance and Evaluation

The Convolutional Neural Network (CNN) model developed for hand sign language detection demonstrated excellent performance during the evaluation phase. After training, the model achieved a test accuracy of approximately 95.29%, as indicated by the final evaluation metrics.

Training Results:

- **Loss:** The model's final loss value was 0.3069, which indicates how well the model is performing in terms of minimizing the error between predicted and actual outputs.
- **Accuracy:** The training accuracy reached **95.29%,** demonstrating the model's high capability in correctly classifying hand sign images.

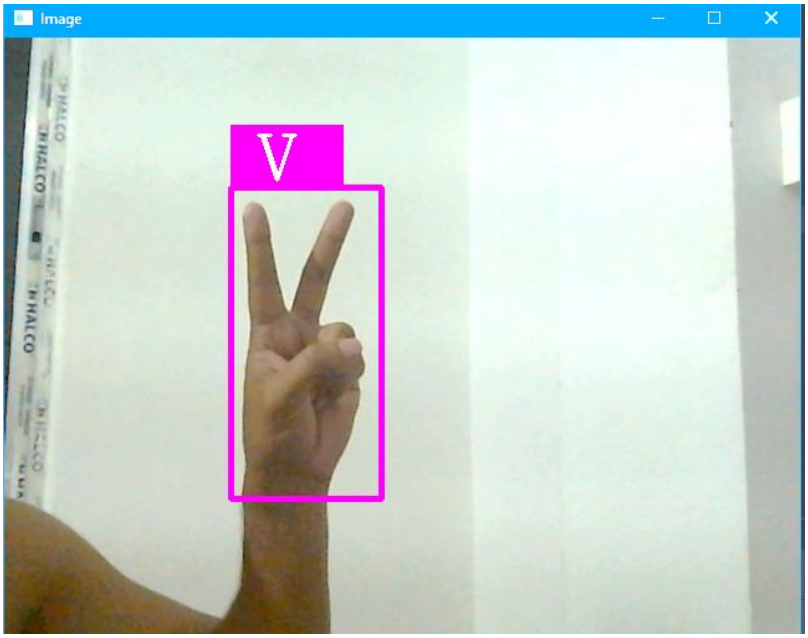
Testing Results:

- **Test Accuracy:** The model achieved an accuracy of **95.29%** on the test set, confirming that it generalizes well to new and unseen data.

Real-Time Prediction

After achieving a high accuracy with the trained CNN model, we moved on to deploying the model for real-time hand sign language detection. By integrating the trained model with OpenCV, we provided vision capabilities to the computer. This involved creating an OpenCV window that captures live video feed from a camera, processes each frame, and uses the model to predict hand gestures in real-time.

The OpenCV window displays the video feed and overlays the predicted hand sign on the screen, allowing users to see the results instantly. This integration not only demonstrates the practical application of our model but also provides an interactive way to test and validate the model's performance in real-world scenarios. By leveraging the powerful combination of TensorFlow for model training and OpenCV for image processing and visualization, we successfully created a comprehensive system for hand sign language recognition.



WEB APPLICATION DEPLOYMENT WITH STREAMLIT

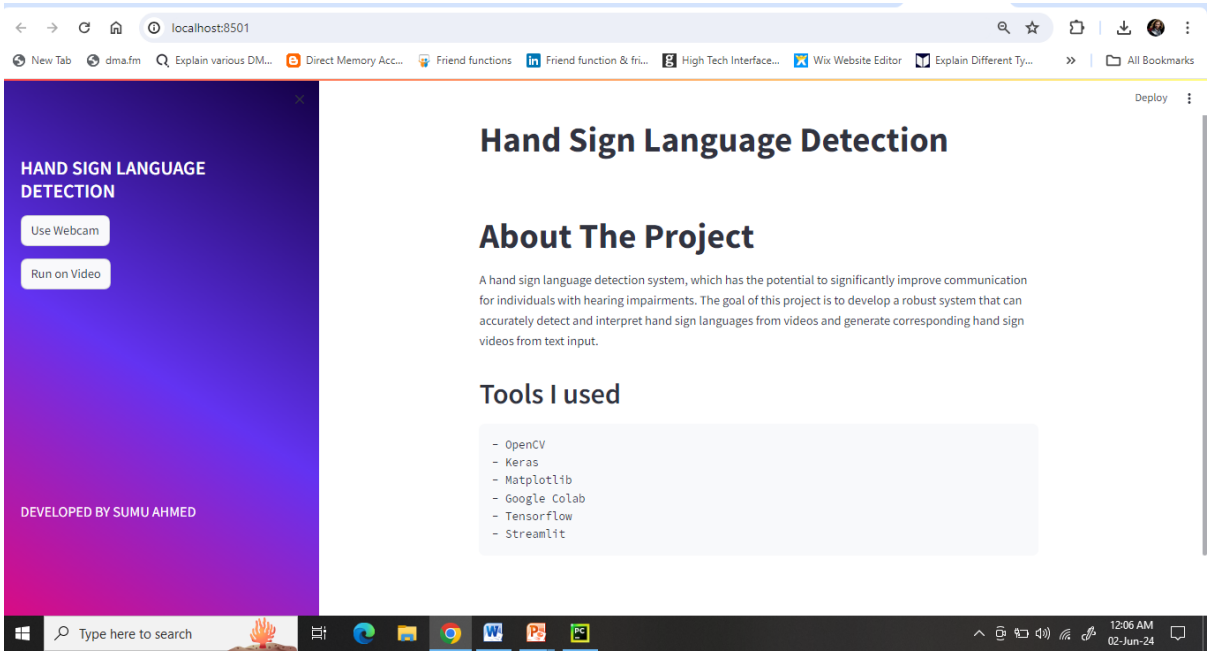
Following the successful integration of real-time hand sign detection using OpenCV, we aimed to make the system more accessible by deploying it as a web application. To achieve this, we utilized Streamlit, a powerful and user-friendly framework for building interactive web applications in Python.

Using Streamlit, we created an intuitive web interface that allows users to interact with the hand sign recognition model through their web browser. The application captures images from the user's webcam, processes them using the trained CNN model, and displays the predicted hand sign on the web page. This setup not only makes the hand sign recognition system widely accessible but also simplifies the user experience by providing a seamless interface for interacting with the model.

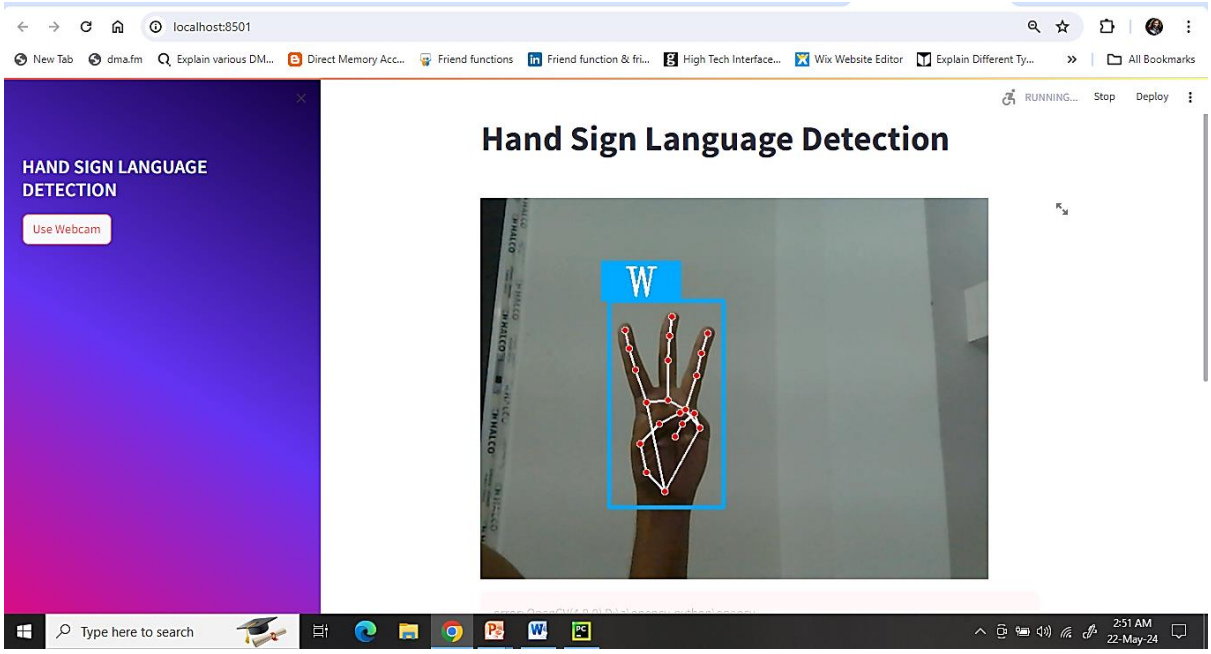
Streamlit's straightforward setup and ease of deployment enabled us to quickly transform our model into a functional web application. This step significantly enhances the usability of our project, allowing users to benefit from real-time hand sign detection directly from their web browsers without the need for complex installations or configurations.

RESULTS

Screenshot 1:



Screenshot 2:



SOURCE CODE

▪ Data Collection

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

cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)

offset = 20
imgSize = 300

folder = "Data/C"
counter = 0

imgCrop = None

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]

    if imgCrop is not None and imgCrop.size > 0:
        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)

```

▪ Model Training

```
"""Define the CNN model"""

model = tf.keras.models.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(img_size, img_size, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Dropout(0.4),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(128, (3, 3), activation='relu'),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(128, activation='relu'),
    tf.keras.layers.Dense(train_generator.num_classes, activation='softmax')
])
model.summary()

"""**Compile the model**"""

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

"""**Train the model using generators**"""

history = model.fit(
    train_generator,
    epochs=30,
    validation_data=validation_generator
)

# Evaluate the model
test_loss, test_acc = model.evaluate(validation_generator)
print(f'Test accuracy: {test_acc}')

```

▪ Prediction

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

cap = cv2.VideoCapture(0)
detector = HandDetector(maxHands=1)
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")

offset = 20
imgSize = 224

folder = "Data/C"
counter = 0

labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V",
          "W", "X", "Y", "Z"]

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, draw=False)
            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)

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

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

```

- **User Interface Code (Streamlit)**

```python
import streamlit as st
import cv2
import numpy as np
import tempfile
import os
from cvzone.HandTrackingModule import HandDetector
from cvzone.ClassificationModule import Classifier
import math

# Initialize the hand detector and classifier
detector = HandDetector(maxHands=1)
classifier = Classifier("Model/keras_model.h5", "Model/labels.txt")
labels = ["A", "B", "C", "D", "E", "F", "G", "H", "I", "J", "K", "L", "M", "N", "O", "P", "Q", "R", "S", "T", "U", "V",
          "W", "X", "Y", "Z"]

# Streamlit app
st.title("Hand Sign Language Detection")

st.markdown(
    """
    <style>
    .main .block-container {
        padding-top: 30px;
    }
    [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {
        width: 350px;
    }
    [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {
        width: 200px;
        margin-left: -300px;
    }
    .copyright-text {
        padding-top: 250px;
        left: 10px;
        font-size: 18px;
        color: #fff;
    }
    .title-text{
        padding-top: 0px;
        color: #fff;
    }
    </style>
    """,
    unsafe_allow_html=True,
)
st.sidebar.markdown('<h1 class="title-text">HAND SIGN LANGUAGE DETECTION</h1>', unsafe_allow_html=True)

st.markdown(
    """
        <style>
        [data-testid="stSidebar"][aria-expanded="true"] > div:first-child {
            width: 400px;
            background: rgb(218,12,131);
            background: linear-gradient(32deg, rgba(218,12,131,1) 0%, rgba(99,51,242,1) 46%, rgba(20,0,71,1) 100%);
        }
        [data-testid="stSidebar"][aria-expanded="false"] > div:first-child {
            width: 400px;
            margin-left: -400px;
        }
        </style>
        """,
    unsafe_allow_html=True,
)

```

```python

def process_webcam():
    cap = cv2.VideoCapture(0)
    stframe = st.empty()

    while cap.isOpened():
        ret, frame = cap.read()
        if not ret:
            break

        hands, img = detector.findHands(frame)
        if hands:
            hand = hands[0]
            x, y, w, h = hand['bbox']
            imgWhite = np.ones((150, 150, 3), np.uint8) * 255
            imgCrop = frame[y - 20:y + h + 20, x - 20:x + w + 20]

            aspectRatio = h / w
            if aspectRatio > 1:
                k = 150 / h
                wCal = math.ceil(k * w)
                imgResize = cv2.resize(imgCrop, (wCal, 150))
                wGap = math.ceil((150 - wCal) / 2)
                imgWhite[:, wGap:wCal + wGap] = imgResize
            else:
                k = 150 / w
                hCal = math.ceil(k * h)
                imgResize = cv2.resize(imgCrop, (150, hCal))
                hGap = math.ceil((150 - hCal) / 2)
                imgWhite[hGap:hCal + hGap, :] = imgResize

            imgWhite = cv2.resize(imgWhite, (150, 150))
            prediction, index = classifier.getPrediction(imgWhite, draw=False)

            cv2.rectangle(frame, (x - 30, y - 20 - 50),(x - 20 + 90, y - 20 - 50 + 50), (255, 170, 2), cv2.FILLED)
            cv2.putText(frame, labels[index], (x, y - 26), cv2.FONT_HERSHEY_COMPLEX, 1.7, (255, 255, 255), 2)
            cv2.rectangle(frame, (x - 20, y - 20), (x + w + 20, y + h + 20), (255, 170, 2), 4)

        stframe.image(frame, channels="BGR")

    cap.release()


if st.sidebar.button('Use Webcam'):
    process_webcam()


    def process_video():
        cap = cv2.VideoCapture(0)
        results = []

        while cap.isOpened():
            ret, frame = cap.read()
            if not ret:
                break

            hands, img = detector.findHands(frame)
```

```
            if hands:
                hand = hands[0]
                x, y, w, h = hand['bbox']
                imgWhite = np.ones((150, 150, 3), np.uint8) * 255
                imgCrop = frame[y - 20:y + h + 20, x - 20:x + w + 20]

                aspectRatio = h / w
                if aspectRatio > 1:
                    k = 150 / h
                    wCal = math.ceil(k * w)
                    imgResize = cv2.resize(imgCrop, (wCal, 150))
                    wGap = math.ceil((150 - wCal) / 2)
                    imgWhite[:, wGap:wCal + wGap] = imgResize
                else:
                    k = 150 / w
                    hCal = math.ceil(k * h)
                    imgResize = cv2.resize(imgCrop, (150, hCal))
                    hGap = math.ceil((150 - hCal) / 2)
                    imgWhite[hGap:hCal + hGap, :] = imgResize

                imgWhite = cv2.resize(imgWhite, (150, 150))
                prediction, index = classifier.getPrediction(imgWhite, draw=False)
                results.append(labels[index])

        cap.release()
        return results

if st.sidebar.button('Run on Video'):
    uploaded_file = st.file_uploader("Choose a video file", type=["mp4", "mov", "avi"])
    if uploaded_file is not None:
        if st.button('Process Uploaded Video'):
            with tempfile.NamedTemporaryFile(delete=False) as tmp_file:
                tmp_file.write(uploaded_file.read())
                tmp_filepath = tmp_file.name

            results = process_video(tmp_filepath)
            os.remove(tmp_filepath)

            st.write("Detection Results:")
            st.write(results)

st.markdown('''
                # About The Project \n
    A hand sign language detection system, which has the potential to significantly improve communication for individuals with hearing impairments. The goal
    of this project is to develop a robust system that can accurately detect and interpret hand sign languages
    from videos and generate corresponding hand sign videos from text input.\n

    ## Tools I used
        - OpenCV
        - Keras
        - Matplotlib
        - Google Colab
        - Tensorflow
        - Streamlit
''')

st.sidebar.markdown('<p class="copyright-text">DEVELOPED BY SUMU AHMED</p>', unsafe_allow_html=True)

```

CONTINUOUS IMPROVEMENT

Continuous improvement is a critical aspect of this project. After deploying the initial version of the hand sign recognition system, we collected user feedback and analyzed the model's performance in real-world scenarios. This feedback loop allowed us to identify areas for enhancement, such as improving the accuracy of specific hand signs, reducing latency, and enhancing the user interface.

We implemented incremental updates to the model by refining the training dataset, optimizing the CNN architecture, and incorporating more robust preprocessing techniques. Regular testing and validation ensured that each update contributed to the overall improvement of the system, maintaining its reliability and effectiveness.

CHALLENGES AND SOLUTIONS

Throughout the development of this hand sign recognition system, we encountered several challenges:

1. **Data Imbalance:** Some hand signs had fewer samples in the training dataset, leading to biased predictions. We addressed this by augmenting the dataset and ensuring a balanced representation of all classes.
2. **Real-Time Performance:** Achieving low latency in real-time predictions was crucial. We optimized the model and utilized efficient image processing techniques in OpenCV to ensure quick and accurate predictions.
3. **User Interface:** Designing an intuitive and user-friendly interface was essential for broad adoption. Using Streamlit, we created a simple yet effective web application that allowed seamless interaction with the model.
4. **Model Generalization:** Ensuring the model's accuracy across different lighting conditions and backgrounds was a challenge. We enhanced the model's robustness by incorporating diverse training data and implementing real-time image normalization techniques.

CONCLUSION

In conclusion, the development and deployment of the hand sign language detection system demonstrate the successful application of Convolutional Neural Networks and modern web technologies. The project highlights the potential for AI to create accessible and impactful solutions for real-world problems, particularly in assisting individuals with hearing impairments.

Through continuous improvement, user feedback, and addressing challenges, we have created a robust and user-friendly system that accurately recognizes hand signs in real-time. This project serves as a foundation for further advancements in gesture recognition and AI-driven communication tools.

REFERENCES

1. Chollet, F. (2017). *Deep Learning with Python*. Manning Publications.
2. Abadi, M., Barham, P., Chen, J., et al. (2016). *TensorFlow: A System for Large-Scale Machine Learning*. OSDI.
3. Howard, A. G., Zhu, M., Chen, B., et al. (2017). *MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications*. arXiv preprint arXiv:1704.04861.
4. OpenCV documentation. Available at: https://docs.opencv.org/
5. Streamlit documentation. Available at: https://docs.streamlit.io/