RSET
RAJAGIRI SCHOOL OF
ENGINEERING & TECHNOLOGY
(AUTONOMOUS)

Project Phase 2 Report On

Interactive Dance Mudra Learning Tool

Submitted in partial fulfillment of the requirements for the award of the degree of

Bachelor of Technology

in

Computer Science and Engineering

By

Nekha S Thomas(U2003149)
Pooja Menon(U2003160)
Sandra Sunil(u2003186)
Sayujya Salim(U2003189)

Under the guidance of

Ms.Amitha Mathew

Department of Computer Science and Engineering
Rajagiri School of Engineering & Technology (Autonomous)
(Parent University: APJ Abdul Kalam Technological University)
Rajagiri Valley, Kakkanad, Kochi, 682039
April 2024

CERTIFICATE

This is to certify that the project report entitled ***"Interactive Dance Mudra Learning Tool"*** *is a bonafide record of the work done by* ***Nekha S Thomas(U2003149), Pooja Menon(U2003160), Sandra Sunil(u2003186) and Sayujya Salim (U2003189)****, submitted to the Rajagiri School of Engineering & Technology (RSET) (Autonomous) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology (B. Tech.) in Computer Science and Engineering during the academic year 2023-2024.*

Ms.Amitha Mathew
Project Guide
Assistant Professor
Dept. of CSE
RSET

Ms.Anita John
Project Coordinator
Assistant Professor
Dept. of CSE
RSET

Dr.Preetha K.G.
Professor & Head of the Department
Dept. of CSE
RSET

ACKNOWLEDGMENT

We wish to express our sincere gratitude towards **Dr.P.S. Sreejith**, Principal of RSET, and **Dr.Preetha K.G.** , Head of the Department of Computer Science for providing us with the opportunity to undertake our project, "Interactive Bharatanatyam Learning Tool".

We are highly indebted to our project coordinator, **Ms.Anita John**, Assistant Professor, Department of Computer Science for her valuable support. We also thank **Mr.Sajanraj T.D.**, Assistant Professor, Department of Computer Science for his valuable support during Phase 1 of our project.

It is indeed our pleasure and a moment of satisfaction for us to express our sincere gratitude to our project guide **Ms.Amitha Mathew** for her patience and all the priceless advice and wisdom she has shared with us.

Last but not the least, we would like to express our sincere gratitude towards all other teachers and friends for their continuous support and constructive ideas.

Nekha S Thomas

Pooja Menon

Sandra Sunil

Sayujya Salim

Abstract

Bharatanatyam is an Indian classical dance, which is composed of various body postures and hand gestures. This ancient art of dance has to be studied under the supervision of experts but at present there is dearth of Bharatanatyam dance experts .Interactive Bharatanatyam Learning Tool aims to bring out the culture and tradition of Bharatanatyam dance form by detecting the mudras and interpreting them using YOLO V5 amd V8. YOLOs excel at image-related tasks due to their ability to automatically learn and extract features from data using bounding boxes.In the world of traditional Indian art forms, Bharatanatyam stands as a vibrant expression of culture and history. To bring this classical dance to enthusiasts worldwide, we introduce an Interactive Bharatanatyam Learning Tool. This innovative platform offers a comprehensive, accessible, and engaging learning experience, transcending geographical boundaries. This tool is not merely a learning platform but a cultural bridge that connects individuals with the grace and beauty of Bharatanatyam, ensuring its legacy for generations to come. The input dataset contains 25 mudras with approximately 200 images in each category. The images were captured under different angles and under the same background. We assume that the images are of the same quality and are taken under the same background. The input given will be a set of images showing the mudras and the output is to interpret the detected mudra along with its name and also to display the corrected mudra in case if what is shown is incorrect.

Contents

List of Abbreviations

ROI- Region of Intrest

UI- User Interface

HTML- Hypertext Markup Language

CSS- Cascading Style Sheets

YOLO- You Only Look Once

ResNet - Residual Neural Network

C3D - 3D ConvNet

LSTM- Long Short term Memory

SeST- Selective Spatiotemporal features learning

mAP- mean Average Precision

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

With its origins in the historic temples of Tamil Nadu, Bharatanatyam is a classical Indian dance style known for its exquisite footwork, elegant facial expressions, and rich narrative. Rich hand gestures (mudras), rhythmic foot dances, and expressive face expressions that narrate stories from Hindu mythology and ancient literature are all part of its repertoire. The dance blends the spiritual and the creative, requiring its practitioners to be precise, flexible, and emotionally invested. Bharatanatyam, traditionally performed by women, has developed over ages, adjusting to new circumstances while preserving its fundamental cultural elements. Its beauty, cultural significance, and ageless depiction of human emotions and tales never fail to enthral audiences.The preservation of Bharatanatyam is imperative in order to protect artistic brilliance, cultural identity, and legacy. This traditional Indian dance style offers a distinctive way to portray history and morals, encapsulating centuries of tradition. By preserving it, a rich legacy is passed down from one generation to the next, promoting cultural continuity and adding to the diversity of the world. Beyond its aesthetic appeal, Bharatanatyam has socially cohesive and educational benefits while acting as a unifying factor in communities. Bharatanatyam's preservation allows for its continuous relevance and enjoyment in modern situations while also paying tribute to the past and spurring creativity. In contemporary times, technology has played a pivotal role in the preservation of culture.Online platforms and digital tools facilitate accessible learning, allowing enthusiasts to engage with the dance form globally.

1.2 Problem Definition

The aim of the project is to develop a web application that leverages computer vision and machine learning techniques to recognize and provide feedback on the correctness

of Bharatanatyam mudras (hand gestures) based on uploaded images. This will assist Bharatanatyam learners and enthusiasts in mastering the art of mudra execution, merging tradition with technology.

1.3 Scope and Motivation

A Bharatanatyam mudra learning application can cover a wide range of topics. Through the use of technology, it is possible to overcome geographical limitations and provide a worldwide platform for aficionados to explore the complex world of Bharatanatyam hand gestures. Both novices and more experienced users can benefit from an interactive, tailored learning experience offered by such an application. Additionally, the application can act as a bridge across cultures, bringing Bharatanatyam to a wider range of viewers and assisting in the preservation of this traditional Indian dance style.
The motivation behind creating a Bharatanatyam mudra learning application is to promote inclusivity and cultural knowledge. It aims to remove obstacles to education so that everyone who is interested can interact with and enjoy this age-old art form, regardless of where they live or their physical limitations. The application is also driven by the educational benefits it provides to institutions and schools, making it possible to incorporate cultural studies into curricula. In addition to creating career opportunities, the application helps create a global community by fusing technology with the artistic expression found in Bharatanatyam. This promotes communication, teamwork, and a common celebration of the rich legacy of Indian classical dance.

1.4 Objectives

The objective of this project is threefold.

1. Mudra Recognition : To develop a system that can accurately recognize and identify Bharatanatyam mudras from images.

2. Correction and Feedback : Once a mudra is recognized, the system should provide feedback and corrections to the user.

3. User-Friendly Interface : Create a user-friendly web application where users can easily interact with the model.

1.5 Challenges

Creating a Bharatanatyam mudra learning application requires resolving technological issues, creating an intuitive user interface, and putting good teaching practices into practice. It is necessary to strike a balance between cultural factors like inclusion and accurate depiction.

1.6 Assumptions

We assume that the images of Bharatanatyam mudras provided by users are of reasonable quality. Also, we assume that the background in the images is relatively consistent and does not introduce excessive noise or complexity. For real time recognition we assume the background is plain and has good lighiting while capturing.

1.7 Societal / Industrial Relevance

The potential for preserving and promoting cultural heritage makes a Bharatanatyam mudra learning tool project socially relevant. The project helps spread ancient art forms by offering lovers worldwide engaging and accessible learning opportunities. Additionally, the application promotes inclusivity by dismantling geographical barriers and opening up Bharatanatyam mudra learning to a wide range of users. This fosters a sense of camaraderie among students who are interested in this traditional Indian dance form in addition to supporting cultural understanding. Furthermore, by advancing cultural studies, supporting cross-cultural understanding, and supporting larger objectives of cultural preservation, the initiative may also be educationally beneficial.

1.8 Organization of the Report

The report is organized into several sections, each of which addresses a specific aspect of the project. Chapter 1 provides an introduction which looks into the background of the project as well as defines the problem. This chapter also includes scope, motivation,

objectives, challenges, assumptions and discusses the societal relevance of the project. Chapter 2 briefs about the different literature available related to the project for comprehensive understanding of the current state of knowledge in the field. In chapter 3 we discuss the hardware , software and functional requirements of the project. Chapter 4 looks into the design of the application as well as the work schedule and module division. The system implementation along with dataset is dicussed in chapter 5. Chapter 6 provides a conclusion and outlines the future extensions possible in the project. The report also includes a list of abbreviations, figures, and tables, as well as a list of publications and appendices containing additional information on the research.

Finally, this chapter provides a thorough overview and foundation for the project by exploring its historical origins, explaining its applicability, and clarifying the particular issues it seeks to solve. This chapter clarifies the difficulties inherent in the project by highlighting important issues. The aims are stated in a way that prepares the reader for a thorough examination in later chapters. The reader gains the underlying knowledge needed to interact with the project's subtleties and importance through this fundamental investigation, opening the door for a more in-depth examination in the ensuing chapters.

Chapter 2

Literature Survey

2.1 Sign Language Recogoniton using Mediapipe and Deep Learning

Mediapipe, developed by Google, includes hand tracking solutions that can be applied to sign language recognition. The Hand Tracking module can detect and track hand movements, which is essential for recognizing sign language gestures. Advances have been made in achieving real-time sign language recognition. The integration of models with efficient architectures allows for practical applications, such as sign language translation systems. Some research explores combining multiple modalities, such as vision and sensors, to improve accuracy in sign language recognition. This research [2] includes the integration of depth information or skeletal tracking in addition to image data through which MediaPipe is used. The architecture used here uses 3 stages with 7 sections. It uses Feed Forward Neural Network. A feedforward neural network is a fundamental type of artificial neural network used in deep learning and machine learning. They are widely used for various tasks, including image recognition, natural language processing, regression, and classification. The dense layers perform matrix-vector multiplication in the background, and the dropout layers reduce the overfitting of the model by modifying outgoing edges of hidden layer neurons randomly and resetting it to 0 at each iteration during the training process. Figure 2.1 shows the Mediapipe Framework using different hand skeletal points as the landmark for different points of the hand as stated in [2]. Glove-based and sensor based inputs for sign language recogonition are shown in [3] and [4]. Smart gloves are used in glove-based SLR architecture to measure hands' location, orientation, velocity, and other characteristics utilizing sensors and microcontrollers. Cameras are used in vision-based SLR techniques to identify hand motions. The majority of computer vision-based SLR systems work by extracting features including gesture recognition, hand form recognition, edge detection, and skin color segmentation. However, the majority of these solutions are

only compatible with platforms with high processing power since they require too much processing power to operate in real-time on low-end computing devices like mobile phones. Moreover, practically all of these methods show the hit and miss nature of hand-tracking techniques.

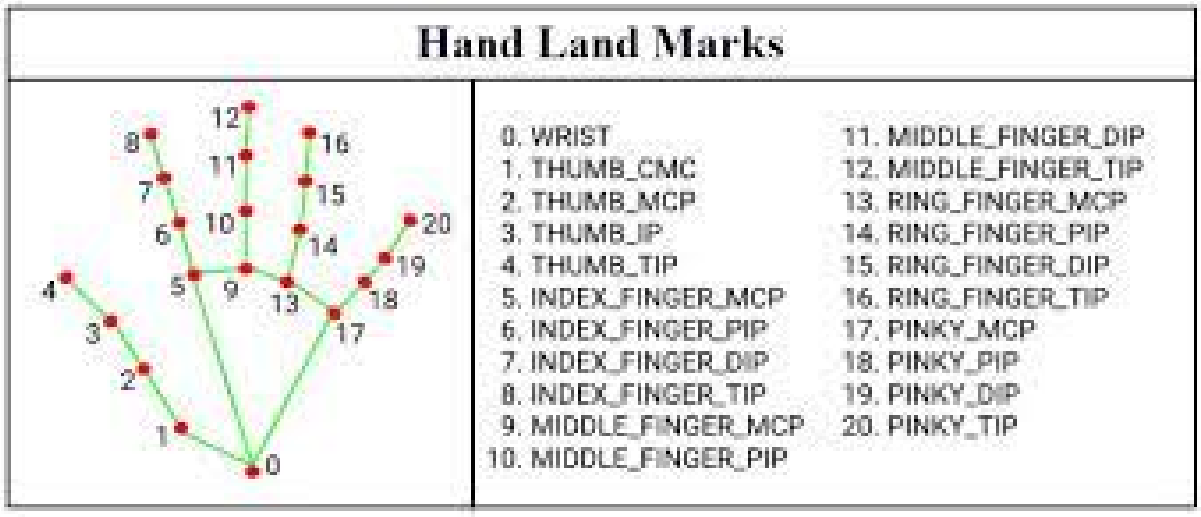


Figure 2.1: MediaPipe Framework [2].

2.2 A Moroccan Sign Language Recognition Algorithm Using a Convolution Neural Network

It represents a model for image processing recognition of gestures, based on artificial neural networks, with a focus on translating sign language into voice/text format.

The proposed model [1] uses a convolutional neural network (CNN) to recognize hand gestures. The system includes a large dataset of static images of Moroccan sign language signals from various angles. The authors explain the system architecture, which is a classical combination of convolution and max pooling. They have chosen a lightweight network to obtain a fast classification allowing real-time classification and localization. The CNN process includes many operations such as convolution, ReLU, pooling, etc. to determine the output by extracting the sign language's information. A database containing hand gestures with meanings is used. It includes a few pictures from our database before they were converted to binary. The results with related works were comapred and achieved the best performance with an accuracy rate of 98.7 percent. It also discussed the metrics used to measure the performance of the classifier, such as precision, recall, F-measurement,

and accuracy. It is a comprehensive study on recognizing Moroccan sign language using a convolutional neural network, providing detailed information on the system architecture, dataset, obtained results, and future directions. The proposed model has the potential to contribute significantly to the one which is immersive game technology, the other is in human–computer interaction, and the interpretation of sign language.

2.3 Hand Gesture Recognition using YOLOv3 Model

The proposed real-time hand gesture recognition system is based on YOLOv3 deep learning model. This system is trained on a dataset of hand gestures labeled in both Pascal VOC and YOLO formats and achieves high accuracy in real-time gesture recognition. The research also discusses practical applications of the system in improving living systems that assist human-computer interaction in both healthy and disabled individuals. In addition, the authors compare the performance of their YOLOv3-based model with different methods and highlight its superior results in accuracy, precision, recall and F-1 scores. The abstract lays the groundwork for an in-depth study of the development and potential implications of real-time hand gesture recognition based on deep learning.

2.3.1 YOLOv3 Model

Using the efficient Darknet-53 architecture and its 53 convolutional layers, YOLOv3 stands out for its intelligent features. Its versatile design allows multiple observations to be made by segmenting the input images, which facilitates the prediction of bounding boxes and class probabilities at different scales. The inclusion of anchor boxes is key in improving the accuracy of bounding box predictions for differently shaped objects. In addition, the Feature Pyramid Network (FPN) enriches YOLOv3 by combining different levels of functions, which meets the challenge of detecting objects of different scales and sizes in an image.

Class prediction is another strength of YOLOv3, as it intelligently determines class probabilities for each bounding box. This adaptability makes YOLOv3 well suited for tasks involving multiple classes of objects, increasing its versatility in various applications. In addition, the model is strategically pre-trained on the COCO dataset, which is an extensive image archive containing 80 different object classes. This pre-training enables

YOLOv3 to effectively generalize and excel in many object recognition tasks.

In addition to its architectural sophistication, YOLOv3 is known for its real-time object detection capability, which achieves a harmonious balance between speed and accuracy. This makes it particularly suitable for scenarios where low latency is most important, such as in the field of autonomous vehicles and real-time tracking systems. In addition, YOLOv3 the open source nature of collaborative innovation, allowing researchers and developers not only to use the model, but also to modify and apply it to different applications. The vibrant community around YOLOv3, along with its availability in multiple programming languages, has contributed greatly to its widespread adoption and continuous improvement.

The evolution of YOLOv3 into subsequent versions such as YOLOv4 and YOLOv5 underscores its continued influence and commitment to improvement. These iterations build on the strengths of their predecessors and push the boundaries of accuracy, speed and additional features. As a result, YOLOv3 and its successors will find applications in various fields, from autonomous vehicles and surveillance to robotics, meeting the demand for real-time object detection and localization with unparalleled versatility and efficiency.

2.4 Unravelling of Convolutional Neural Networks through Bharatanatyam Mudra Classification with Limited Data

The methodology encompasses various critical stages for the classification of single-hand Bharatanatyam mudras using CNNs. The process begins with the creation of the mudra dataset, involving data acquisition, preprocessing, cleansing, and augmentation. The dataset is prepared to ensure its quality and diversity.

A pivotal component of the methodology involves the design and utilization of a convolutional siamese neural network, specifically tailored for data cleansing purposes. This architecture proves beneficial for addressing one-shot recognition problems, contributing to the overall robustness of the classification pipeline.

Subsequently, the study explores the application of CNNs for image classification, with a specific focus on the intricate task of classifying single-hand gestures in the context of Bharatanatyam. The authors underscore the efficiency of CNNs in autonomously learning features, eliminating the need for manual intervention compared to the methods in [13],

[14], and [15]. Challenges associated with dynamic backgrounds in the image dataset are also acknowledged and addressed within this stage of the method.

In addition to singular CNN models, the authors elaborate on the development of ensemble models and specialized models. Transfer learning techniques play a significant role in this phase, where pre-trained models undergo fine-tuning on the mudra dataset. The impact of hyperparameter optimization, transfer learning, and ensemble models on the classification accuracy of the developed models is thoroughly examined, providing valuable insights into the effectiveness of these strategies.

To enhance the dataset's size and diversity, the authors incorporate data augmentation techniques. This augmentation step contributes to the overall generalizability of the models, allowing them to perform effectively on a broader range of inputs.

In summary, the method proposed in [12] involves a multi-stage approach that capitalizes on advanced deep learning techniques and shows its efficiency over [16]. The integration of convolutional siamese neural networks, transfer learning, ensemble models, and data augmentation collectively aims to achieve high classification accuracy for Bharatanatyam mudra classification tasks.

2.5 Selective spatiotemporal features learning for dynamic gesture recognition

The method introduces a novel deep learning model called Selective Spatiotemporal features learning (SeST) for dynamic gesture recognition. The SeST model combines the ResC3D network[10] and Convolutional LSTM (ConvLSTM) with a dynamic select mechanism to simultaneously learn short-term and long-term spatiotemporal features, which are complementary to each other. The SeST block enables the ResC3D network and ConvLSTM to adaptively adjust their contributions to classification during feature learning with soft-attention. The SeST model consists of three main components: the ResC3D network, the ConvLSTM network, and the SeST block. The ResC3D network is used to extract local spatiotemporal features, while the ConvLSTM network is used to capture long-range temporal dependencies between features. The SeST block is used to collaboratively learn local and global spatiotemporal features using ResC3D and ConvLSTM at the layer level. The SeST block allows the network to attend to either short-term or

long-term spatiotemporal features on demand, which greatly increases the adaptability of the model.

The proposed SeST model is evaluated on three publicly available datasets: the Sheffield Kinect Gesture (SKIG) dataset, the ChaLearn LAP large scale isolated gesture dataset (IsoGD), and the EgoGesture dataset. The experimental results show that the proposed SeST model outperforms other state-of-the-art methods on all three datasets. The SeST model achieves an accuracy of 98.3% on the SKIG dataset, 94.5% on the IsoGD dataset, and 96.5% on the EgoGesture dataset. The SeST model is also an end-to-end model, which can be embedded in many intelligent system applications.

2.6 Summary and Gaps Identified

Table 2.1: Summary of different methods.

Method	Advantage	Disadvantage
A Moroccan Sign Language Recognition Algorithm Using a Convolution Neural Network [1]	One advantage of the proposed method is its high accuracy rate of 98.7%. This is a significant improvement compared to other state-of-the-art models.	One disadvantage of this method is that it requires a large dataset for training the model . This can be time-consuming and resource-intensive, especially if the dataset needs to be manually labeled.
Sign Language Recognition using Mediapipe and Deep Learning [2]	MediaPipe offers a user-friendly API and pre-trained models, making it accessible for developers with varying levels of expertise. The modular design simplifies the integration of features into applications.	It may not be as customizable as building a solution from scratch, especially for highly specialized or domain-specific requirements.
Hand Gesture Recognition using YOLOv3 Model [17]	YOLOv3 has real-time processing capabilities, making it well suited for applications such as video surveillance, and its ability to handle different types of objects with high accuracy.	It has limitations in accurate localization, struggling with small objects, and its dependence on a large and diverse training dataset.
Unravelling of Convolutional Neural Networks through Bharatanatyam Mudra Classification with Limited Data [12]	Its ability to automatically learn features from the data without manual intervention.	The potential requirement for a large amount of data for effective leveraging of machine learning techniques.
Selective Spatiotemporal Features Learning For Dynamic Gesture Recognition [7]	Adapts ResC3D and ConvLSTM with a dynamic select mechanism for simultaneous short-term and long-term spatiotemporal feature learning.	Highly complex and time consuming.

The gaps identified are:

- Limited Data and Diversity: [5], [7], [12] and [20] highlight the importance of datasets in training their models. However, there is a common challenge of limited data, especially in the context of Bharatanatyam mudra classification and Moroccan sign language recognition [1]. The scarcity of diverse datasets may impact the models' generalizability and performance in real-world scenarios. Future work could focus on creating larger and more diverse datasets for these specific domains.

- Real-time Application Challenges: While the researches discuss real-time applications, they may not delve deeply into the challenges associated with achieving real-time performance[7]. For instance, the computational intensity of some models, such as YOLOv3 [17], might be a bottleneck for deployment on resource-constrained devices. Future research could explore optimization techniques and hardware accelerations to enhance the real-time efficiency of these systems.

- Interactivity and User Feedback: The researches generally focus on the technical aspects of model development and accuracy metrics. However, there is limited discussion on user interaction and feedback mechanisms.

- Cross-Modal Integration: While some researches touch upon combining multiple modalities for improved accuracy[7],[10], there is room for further exploration in seamlessly integrating vision and sensor-based information. For example, exploring the integration of depth information or skeletal tracking alongside image data in MediaPipe-based systems as said in research [2] could enhance the robustness of the models.

- Adaptability to Variations: The researches often focus on specific scenarios, such as recognizing static hand gestures or single-hand Bharatanatyam mudras. Exploring the adaptability of the models to variations in lighting conditions, diverse hand shapes, and complex dance movements could be a fruitful area for future research. This could involve investigating more advanced augmentation techniques or developing models that can handle dynamic[7] and complex gestures.

In conclusion, the presented researches collectively contribute to the advancement of gesture recognition and sign language translation, showcasing the interdisciplinary nature

of this field that intersects technology, dance, and communication. The integration of technologies like MediaPipe and YOLOv3, alongside innovative deep learning models, demonstrates the potential for practical applications in diverse domains.

The exploration of sign language recognition using hand tracking in MediaPipe in [2] highlights the promise of real-time applications, particularly in the context of sign language translation systems. The adaptability of MediaPipe's hand tracking module for sign language recognition underscores its potential impact on fostering inclusive communication for the hearing-impaired.

The use of YOLOv3 for real-time hand gesture recognition further exemplifies the intersection of computer vision and deep learning in enhancing human-computer interaction. YOLOv3's efficiency in balancing accuracy and speed positions it as a valuable tool for applications requiring low-latency object detection, such as robotics and surveillance.

The application of convolutional neural networks (CNNs) to recognize Bharatanatyam mudras introduces the cultural dimension to gesture recognition. Despite challenges such as limited data and dynamic backgrounds, the utilization of CNNs proves effective in autonomously learning features critical for the classification of intricate hand gestures in the context of traditional dance forms.

The SeST model's introduction in dynamic gesture recognition showcases the continual evolution of deep learning architectures to address complex tasks. The incorporation of ResC3D and ConvLSTM networks with a dynamic select mechanism illustrates the significance of adaptability in learning both short-term and long-term spatiotemporal features, presenting a solution for more accurate dynamic gesture recognition.

However, some common gaps across the researches include the challenges associated with limited and homogeneous datasets, the need for real-time efficiency in deployed systems, and the importance of user-centric design principles in interactive applications. Ethical considerations, especially in the representation of cultural elements, also emerge as areas for further exploration.

In essence, these research endeavors collectively propel the field of gesture recognition forward, offering insights, methodologies, and technological solutions that have the potential to impact diverse domains, from dance education to sign language translation, robotics, and human-computer interaction.

Chapter 3

Requirements

3.1 Hardware and Software Requirements

1. Device with Camera Unit:

   A device with a built-in camera or an external webcam is essential for capturing images or video frames of hand gestures. The camera's resolution and quality will impact the clarity of the input data, influencing the model's accuracy.

2. NVIDIA GPU:

   An NVIDIA Graphics Processing Unit (GPU) is highly advantageous for machine learning tasks, especially those involving deep learning frameworks like TensorFlow. GPUs accelerate the training of complex models, significantly reducing processing time compared to using only the CPU. Ensure that the GPU supports CUDA, a parallel computing architecture developed by NVIDIA.

3. Processor - Intel i5:

   A processor with sufficient processing power is crucial for efficient data handling and model computation. An Intel Core i5 processor is a good choice, providing a balance between performance and cost. The processor's speed and the number of cores contribute to the overall system performance.

4. RAM - 8GB:

   Random Access Memory (RAM) is vital for temporarily storing data that the processor actively uses. For a hand mudra classification project, 8GB of RAM is a reasonable starting point. Sufficient RAM ensures smooth execution of code and prevents slowdowns when working with large datasets or complex models.

5. Python (3.7.4):

   Python is a programming language used to develop machine learning. Ensure compatibility with required libraries. Recommended version: 3.7.4.

6. Jupyter Notebook:

   Jupyter Notebook is an interactive development environment commonly used in data science and machine learning. Enables easy experimentation and visualization.

7. TensorFlow (version 2.0.0):

   TensorFlow is an open source machine learning library developed by Google. Ensure compatibility with Python version. Installation: pip install tensorflow==2.0.0 .

8. HTML CSS:

   HTML and CSS can be used to create a simple web-based user interface or to present the results of projects.

9. OpenCV (version 3.4.2):

   OpenCV is a library of computer vision tasks that can be used for image processing. Installation: pip install opencv-python==3.4.2

10. Windows 11:

    The operating system on which you intend to run your machine learning code, the system meets the minimum hardware and software requirements specified for Windows 11.

3.2 Functional Requirements

1. User authentication:

   Ensure secure access to the system by implementing user authentication mechanisms, allowing only authorized users to upload Bharathanatyam hand mudra images.

2. Hand Mudra Images:

Users should be able to easily upload Bharathanatyam hand mudra images to the system, providing a seamless input process for evaluation and corrective operations.

3. Automatic classification:

   Use machine learning algorithms to automatically classify uploaded hand mudra images, simplifying the process of identifying and classifying different Bharathanatyam gestures.

4. Suggestions for improvement:

   Generate correction suggestions for each classified hand mudra based on predefined criteria or rules, helping users refine their hand gestures.

5. User feedback mechanism:

   Implement a feedback mechanism that allows users to provide feedback on the accuracy of the system and classification and the effectiveness of proposed improvements.

Chapter 4

System Architecture

The system architecture of an interactive Bharatanatyam learning tool plays a critical role in facilitating a seamless and effective learning experience for users. The Interactive Bharatanatyam Learning Tool is designed to leverage technology to aid in the learning of Bharatanatyam, a traditional Indian dance form. The architecture encompasses various modules to handle data, training, user interaction, and feedback. The functionality of identifying mudras based on user input through the trained model is a crucial step. The timeline indicates that the training and tuning process is planned from January to March 2024, ensuring a comprehensive training period. The integration of HTML and CSS for the user interface, coupled with the correction mechanism that analyzes joint angles, adds an interactive and corrective dimension to the tool. The system aims to provide users with corrected mudras and associated descriptions for enhanced learning. The Gantt chart provides a clear visual representation of the project timeline, showcasing the systematic progression from November 2023 to the end of March 2024. Milestones such as UI implementation, data pre-processing, architecture building, training, and testing are well-defined.

4.1 System Overview

The Interactive Bharatanatyam Learning Tool is a sophisticated system designed to bridge the realms of traditional Indian dance education and modern technology, offering users an immersive and interactive learning experience. The system's architecture is multifaceted, comprising several interconnected modules to facilitate efficient data processing, training, user interaction, and real-time feedback. The data collection and pre-processing module curates a diverse dataset of Bharatanatyam dance movements, employing image and video segmentation techniques to extract key features and gestures. The training and

model development module harnesses deep learning, particularly YOLOv5 and YOLOv8, to interpret and recognize Bharatanatyam mudras and movements. The user interface, built using web technologies like HTML and CSS, provides an aesthetically pleasing and user-friendly platform for learners. The gesture recognition and feedback module processes real-time user inputs, captured through a camera or input device, enabling the model to provide immediate feedback on the accuracy and correctness of performed Bharatanatyam mudras. The learning path and curriculum module ensures a structured and personalized learning journey for users, while the correction and feedback module offers detailed assessments of the user's performance, guiding them towards improvement in posture, hand movements, and expressions. The seamless integration of these modules ensures a continuous and cyclical learning experience, enhancing users' Bharatanatyam skills progressively. The system utilizes a combination of computer vision libraries (such as OpenCV), deep learning frameworks (like TensorFlow or PyTorch), and potentially augmented reality for an enriched learning experience. Scalability is a key consideration, allowing for the incorporation of additional dance styles, features, and improvements over time, while regular updates based on user feedback contribute to the tool's ongoing effectiveness and relevance. Overall, the Interactive Bharatanatyam Learning Tool stands at the intersection of cultural heritage and technological innovation, redefining the approach to traditional dance education.In figure 4.1, the architecture diagram is displayed.

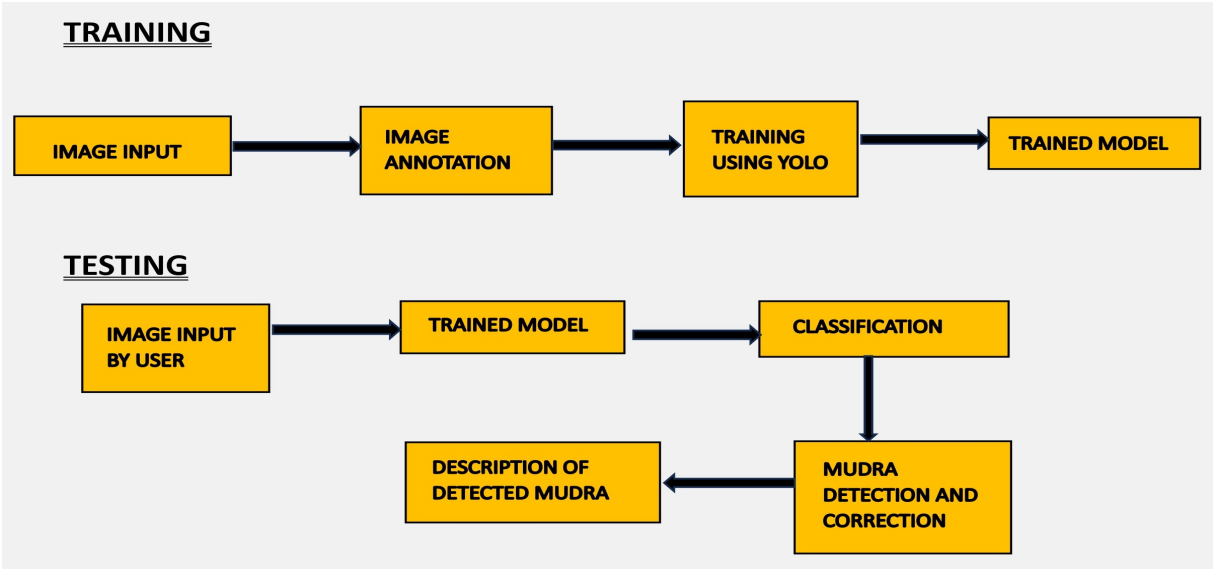


Figure 4.1: Architecture Diagram for Interactive Bharatanatyam Learning tool.

4.2 Architectural Design

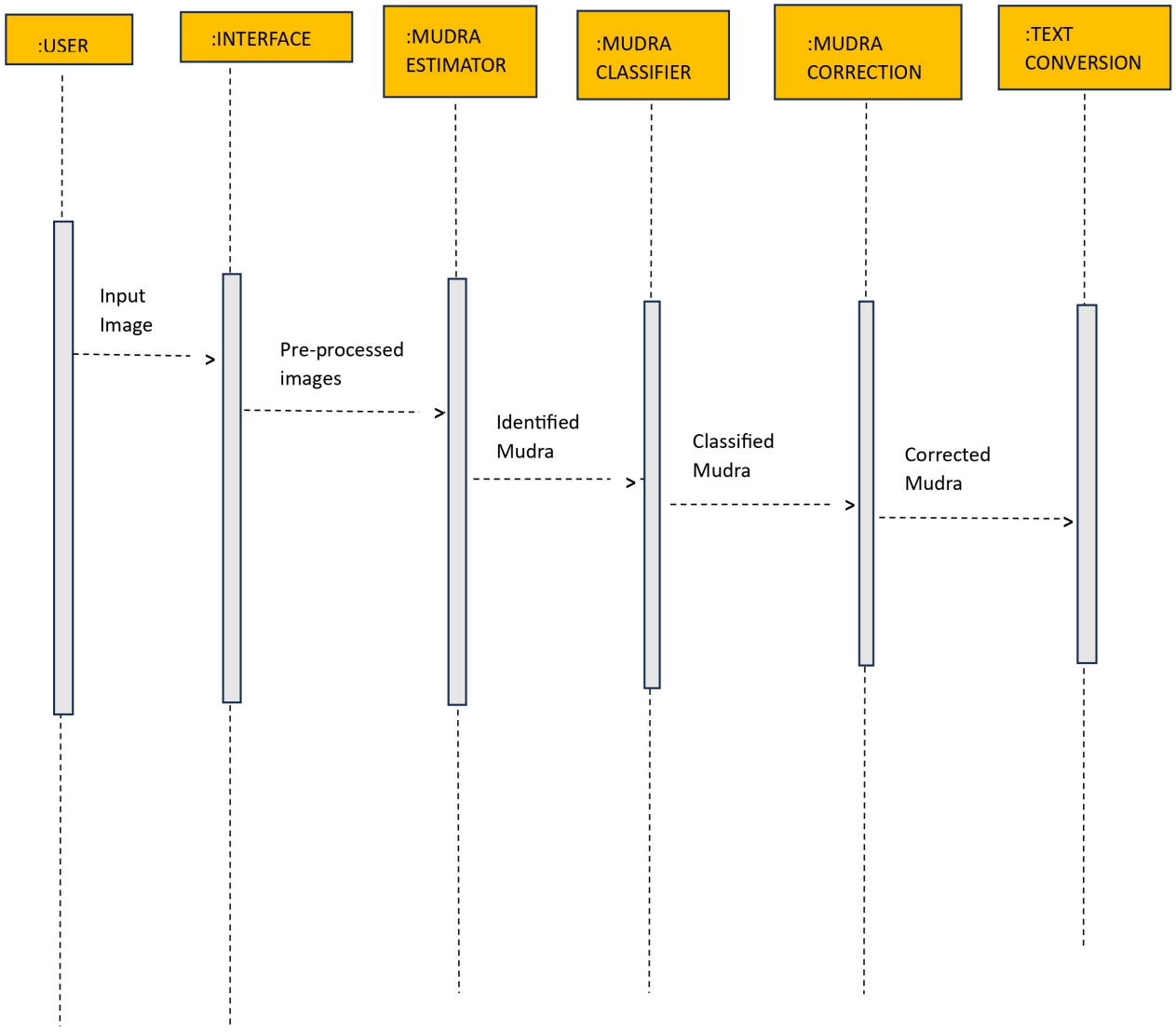


Figure 4.2: Sequence diagram for Interactive Bharatanatyam Learning tool.

4.3 YOLOv5 Architecture

When it comes to object detection models, there are typically two categories: two-stage object detectors and single-stage object detectors. Single-stage detectors, such as YOLO, typically consist of three key components: the Backbone, Neck, and Head, facilitating dense predictions.

1. Model Backbone

   The backbone serves as a pre-trained network designed to extract rich feature representations from images. This process effectively reduces the spatial resolution of the image while enhancing its feature resolution.

2. Model Neck

   The neck component is responsible for extracting feature pyramids, enabling the model to effectively handle objects of varying sizes and scales.

3. Model Head

   The head component carries out the final stage operations. It applies anchor boxes on feature maps and generates the ultimate output, including class predictions, objectness scores, and bounding box coordinates.

In the case of YOLOv5, the architecture follows a similar structure[21] with three main components as shown in Figure 4.3:

1. Backbone:

   YOLOv5 utilizes CSP-Darknet53 as its backbone. CSP-Darknet53 is essentially the Darknet53 convolutional network, the backbone used in YOLOv3, enhanced with the Cross Stage Partial (CSP) network strategy. This strategy preserves the advantageous feature reuse characteristics of DenseNet while mitigating redundant gradient information by truncating the gradient flow.

2. Neck:

   YOLOv5 introduces modifications in its neck component, incorporating a variant of Spatial Pyramid Pooling (SPP) and adapting the Path Aggregation Network (PANet) with BottleNeckCSP in its architecture. PANet, previously used in YOLOv4, facilitates improved information flow and assists in accurate pixel localization for tasks such as mask prediction. In YOLOv5, CSPNet strategy is applied to PANet, enhancing its performance.

3. Head:

   YOLOv5's head consists of three convolution layers responsible for predicting bounding box locations (x, y, height, width), object scores, and class labels.

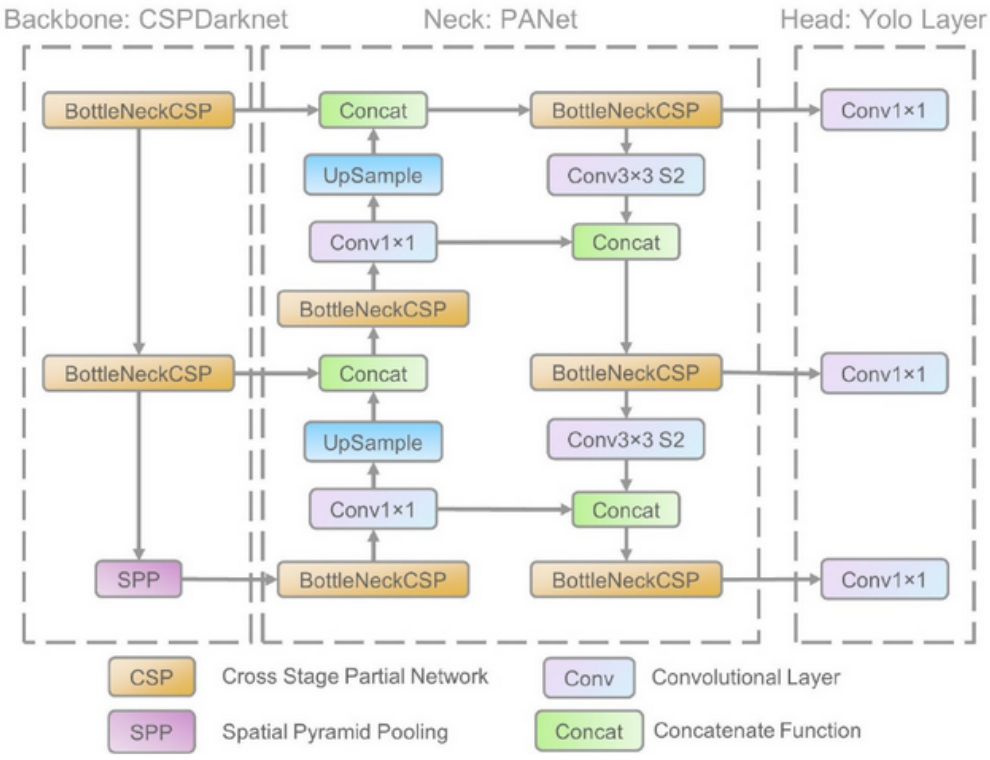


Figure 4.3: Network architecture for YOLO v5 [21].

YOLOv5's modifications aim to enhance performance while optimizing computational efficiency. The utilization of CSPNet strategy and enhancements in the neck component contribute to reducing parameters and computational load, ultimately improving inference speed, a critical factor in real-time object detection models.

4.4 YOLOv8 Architecture

YOLOv8 is an anchor-free model. This means it predicts directly the center of an object instead of the offset from a known anchor box. Anchor boxes were a notoriously tricky part of earlier YOLO models, since they may represent the distribution of the target benchmark's boxes but not the distribution of the custom dataset. Anchor free detection reduces the number of box predictions, which speeds up Non-Maximum Suppression (NMS), a complicated post processing step that sifts through candidate detections after inference. The YOLOV8 Cf2 model [22] is depicted in Figure 4.4.

In the neck component of YOLOv5, significant changes have been made, including replacing the first convolution in the stem with a 3x3 kernel, altering the main build-

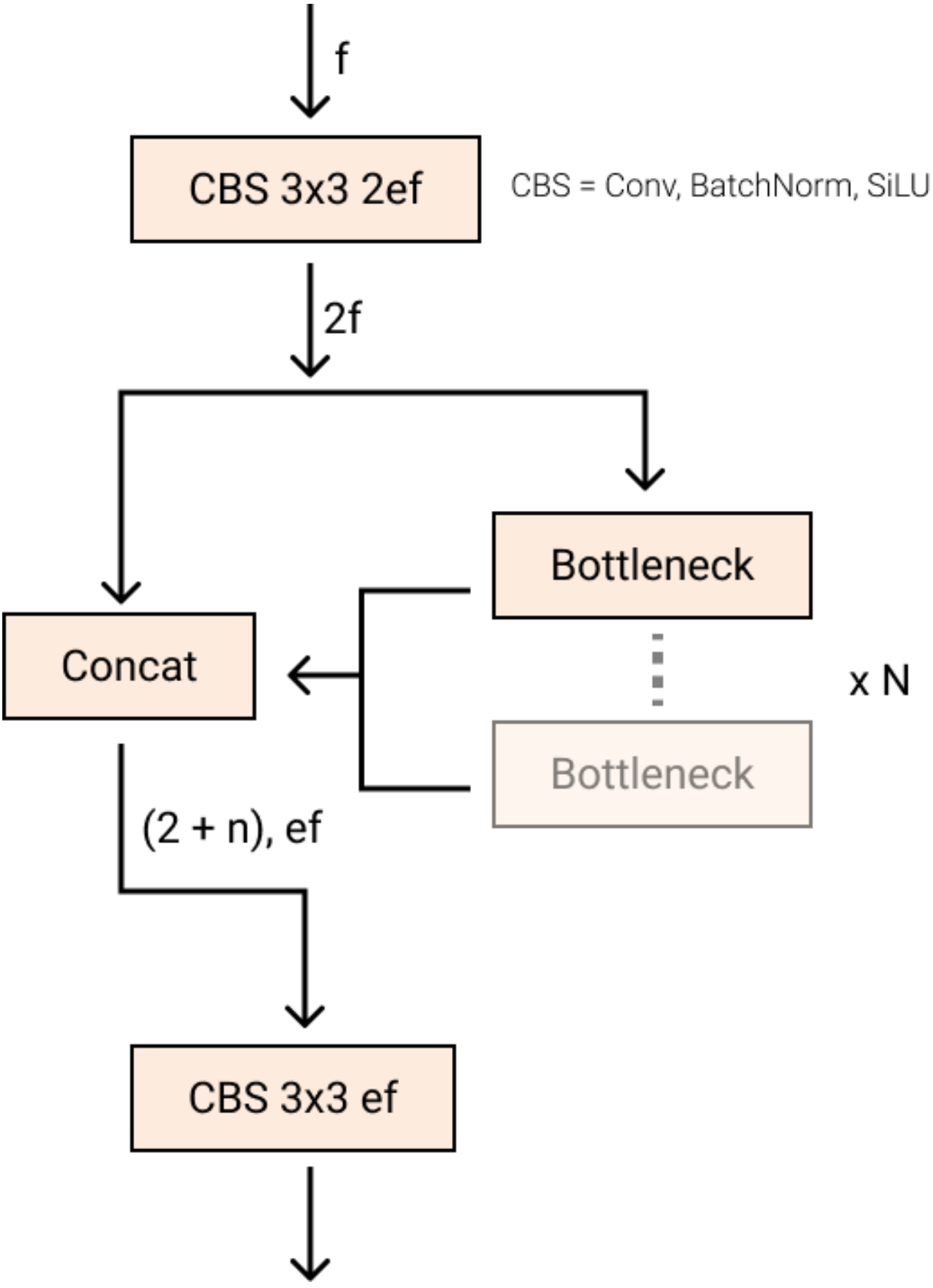


Figure 4.4: YOLOv8 C2f model [22].

ing block, and substituting C2f with C3, where "f" denotes the number of features, "e" represents the expansion rate, and CBS denotes a block comprising a convolution, batch normalization, and SiLU activation. In C2f, all outputs from the Bottleneck are concatenated, while in C3, only the output of the last Bottleneck is utilized. The Bottleneck remains consistent with YOLOv5, but the kernel size of the first convolution has been modified from 1x1 to 3x3. This module's structure is depicted in the accompanying picture. Notably, in the neck, features are directly concatenated without enforcing identical channel dimensions, leading to a reduction in parameter count and the overall size of tensors.

4.5 Module Division

The project has mainly 4 modules as follows:

1. Data annotation.
2. Training And Evaluation.
3. Mudra Identification.
4. User interface and Mudra Correction.

4.5.1 Data annotation

In this module, the dataset for the 25 classes of Bharatanatyam hand mudras is collected. Each image in the dataset is manually annotated by drawing a bounding box around the hand mudra present in the image. Additionally, each image is labeled with its corresponding class, representing the specific hand mudra being depicted. Roboflow provides options for augmentation and segmentation. The final output of this module is a dataset version containing annotated images with bounding boxes and class labels.

4.5.2 Training and Evaluvation

In the training and testing module, the annotated images are used to train a YOLO model. Specifically, YOLOv5 is chosen for static images, while YOLOv8 is selected for real-time detection scenarios. The training process involves feeding the annotated images

into the YOLO model, which learns to detect and classify the hand mudras based on the provided annotations. After training, the model is evaluated to assess its performance in accurately detecting and identifying the hand mudras. The final output of this module is a trained YOLO model capable of detecting Bharatanatyam hand mudras.

4.5.3 Mudra Identification

In this module, either an image containing a hand mudra or a live feed from a laptop camera displaying a hand mudra is provided as input. This input is then passed through the trained YOLO model, which performs mudra identification. The model detects and identifies the hand mudra present in the input image or video feed, providing the corresponding classification.

4.5.4 User Interface and Mudra Correction

The identified hand mudras' bounding boxes are displayed in the user interface, providing visual feedback on the accuracy of the identified mudras. Additionally, the interface provide information and descriptions about the detected mudras, enhancing user understanding and engagement. The final output of this module includes the corrected mudra identifications and accompanying descriptions.

4.6 Work Schedule - Gantt Chart

The work on the project started in November 2023. The UI and design implementation and the data pre-processing such as image segmentation and data splitting to testing ,training and validation was completed by beginning of December 2023.Further the architecture was build in basic CNN, YOLO and VGG16 by the end of December 2023.The training and tuning started in January 2024 and was completed by the mid of March 2024.The testing and evaluation was also been completed by the end of March 2024.The detailed Gantt chart is shown in Figure 4.3.

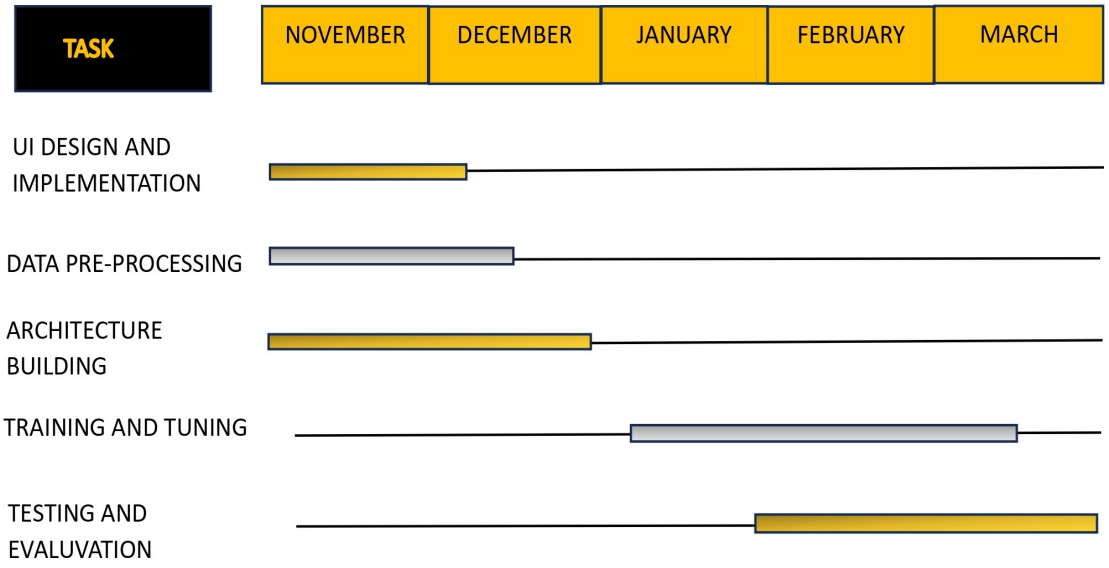


Figure 4.5: Gantt chart for Dance Mudra Learning Tool.

4.7 Budget

The project is completely a software project because of which no budget has been estimated. Although there has been cost involved in purchasing Google Colab Pro version which was required for training the various models.

4.8 Performance Metrics

According to the research [1], there are generally four evaluation measures used. These are accuracy, recall, precision and F-measure. The confusion matrices corresponding to false negative (FN), true negative (TN), true positive (TP) and false positive (FP) are used to measure the above parameters of the classifier. The following formulas define each of the metrics:

TP: is a positive example that correctly predicts the positive class.

TN: refers to the negative that is properly characterized as negative.

FP: refers to a negative example that is incorrectly characterized as positive.

FN: refers to the positive example that is imperfectly characterized as negative.

Precision: precision is a measure of the accuracy of detection.

$$Precision = \frac{TP}{TP + FP}$$

Recall: recall defines that the number of positives recognized correctly.

$$Recall = \frac{TP}{TP + FN}$$

F-measurement: the F-measurement defines the voice means of recall and precision.It is calculated by the formula below which takes both false positives and false negatives into account using the following equation.

$$F - measure = \frac{2TP}{2TP + FP + FN}$$

Accuracy: accuracy is a measure of the accuracy of the detection.

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN}$$

The outlined project on the Interactive Bharatanatyam Learning tool presents a comprehensive approach to integrating technology and dance education. The project is systematically structured into four key modules: Data Pre-processing and Segmentation, Training and Evaluation, Mudra Identification, and User Interface and Mudra Correction.

In conclusion, the project exhibits a thoughtful and structured approach to developing an Interactive Bharatanatyam Learning tool. The integration of technology with dance education holds great potential for enhancing learning experiences. To ensure success, continuous monitoring, testing, and refinement based on user feedback will be essential. The outlined timeline in the Gantt chart appears well-structured, allowing for a step-by-step progression through each project phase. The success of the project will be contingent on the effective integration of the individual modules and the seamless functionality of the user interface and mudra correction mechanisms.

Chapter 5

System Implementation

Interactive Dance Mudra Learning tool is implemented using basic CNN and other algorithms to compare the accuracy of different methodologies in predicting the correct mudras. It has basically 4 modules starting from the data pre-processing and segmentation to the training and evaluation,mudra identification and user interface with mudra correction. The tool identifies the mudras with the help of CNN algorithm and suggest correction based on the mudra shown by the user. It can be live images or videos.

5.1 Datasets Identified

The data set consits of approximately 5000 images under 25 classes of mudras. These are colour images of different mudras taken under a green screen background by different people at different angles. The sample of an image is shown in Figure 5.1 . These

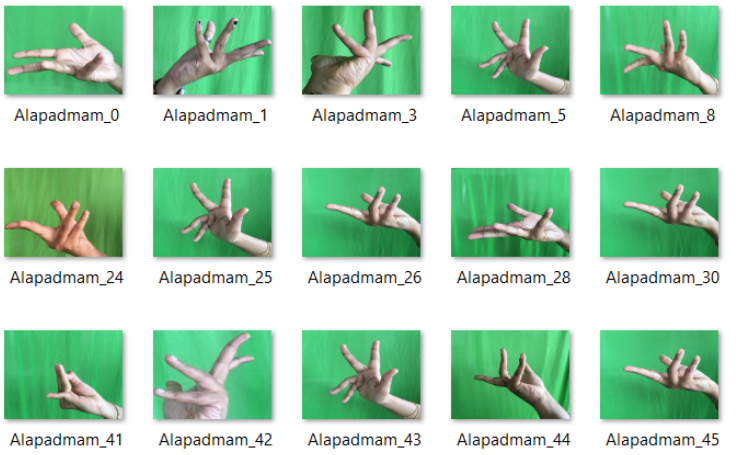


Figure 5.1: Image samples from dataset.

images were segmented to binary images using Python, OpenCV and TensorFlow as shown in Figure 5.2 . This was done by setting a threshold so that the images are clearly

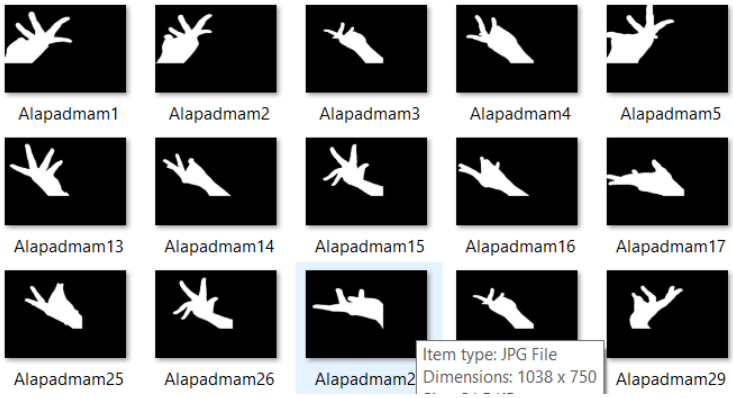


Figure 5.2: Segmented Images into binary.

distinguished from the background. The mudras are in white and the background is in black. The dataset was also split into testing training and validation data of ratios 70 percent, 15 percent and 15 percent respectively of the 27,731 images in each class.

5.2 Proposed Methodology/Algorithms

5.2.1 Data Collection and Selection

A dataset of 27,131 images was gathered from sources on the internet, depicting a wide range of dance mudras in different contexts and lighting conditions. To optimize training efficiency and address resource constraints, the dataset was reduced and refined as follows:

Class Selection: Out of 50 potential classes representing different dance mudras, 25 relevant classes were chosen.

Image Selection: A subset of 200 images was selected for each of the 25 chosen classes, resulting in a curated dataset of 5,000 images.

5.2.2 Annotation

The selected images were annotated using Roboflow, a robust annotation tool, to label the regions containing the dance mudras with bounding boxes. Each annotation included the corresponding class label identifying the detected mudra. This meticulous annotation process ensures that the machine learning models can accurately recognize and classify dance mudras based on the annotated training data.

5.2.3 Preprocessing

During the preprocessing phase, data augmentation techniques were applied to enhance the diversity and robustness of the dataset. Specifically, rotation and shear transformations were employed on the images, effectively expanding the dataset size from 5,000 to approximately 11,000 images.

Rotation: Images were rotated by varying degrees (e.g., 90°, 180°, 270°) to simulate different orientations and viewpoints of the dance mudras.

Shear: Shear transformations were applied to distort the images along the x-axis or y-axis, introducing additional variations in shape and perspective.

By augmenting the dataset through rotation and shear transformations, the goal was to improve the model's ability to generalize and perform well across different orientations and variations of the dance mudras during training. This augmented dataset is essential for training more robust and adaptable machine learning models.

5.2.4 Model selection

In the process of developing the Mudra Detection Tool, various convolutional neural network (CNN) architectures were experimented with to identify the most suitable model for gesture recognition, particularly focusing on integration with a web application's user interface. The trained models included VGG16, a basic CNN architecture, YOLOv5, and YOLOv8.

Among these models, YOLOv5 and YOLOv8 particularly stood out for their capability to provide real-time detections, which is a critical requirement for the application. The YOLO (You Only Look Once) architecture is known for its efficiency in object detection tasks and has been adapted for gesture recognition in this context. YOLO models use a single neural network to predict bounding boxes and class probabilities directly from full images in one evaluation, making them ideal for quick and accurate real-time predictions.

While YOLOv5 and YOLOv8 excelled in real-time performance, other models like VGG16 and a basic CNN were also evaluated for comparison. VGG16, a widely used deep CNN architecture, demonstrated strong performance in image classification tasks but was comparatively slower for real-time gesture detection, making it less suitable for integration into a web application's interface where responsiveness is crucial.

Based on these evaluations, YOLO models were chosen to be integrated into the Mudra Detection Tool due to its balance of accuracy and real-time performance. YOLO offers a good compromise between model complexity and inference speed, allowing for efficient processing of hand gesture recognition within a web application environment. This decision ensures that the tool can deliver responsive and accurate Mudra detection capabilities to end users interacting with the web application.

5.2.5 Model Training

VGG16: Training VGG16 involved leveraging its pre-trained weights on large image classification datasets like ImageNet. This approach provided a solid foundation for learning hierarchical features within the Mudra dataset. The advantages of VGG16 include its well-established architecture and strong performance in image classification tasks. However, training VGG16 from scratch for gesture recognition posed challenges in terms of computational resources and training time due to its deep architecture with a high number of parameters. Fine-tuning and adapting VGG16 for gesture recognition required careful adjustments to balance between performance and efficiency.

YOLOv5 and YOLOv8: Training YOLOv5 and YOLOv8 for gesture recognition provided the advantage of real-time performance and efficient object detection. These models excel in detecting objects with a single forward pass through the network, making them suitable for applications requiring quick response times like real-time gesture recognition. The YOLO architecture's challenge lies in balancing speed and accuracy. While YOLO models are efficient, achieving high accuracy on specific gesture classes required fine-tuning the model architecture, optimizing anchor box priors, and carefully tuning training parameters such as batch size and learning rate. Additionally, integrating YOLOv5 and YOLOv8 into the tool's web application interface necessitated optimizations to ensure seamless deployment and responsiveness in real-world usage scenarios.

5.2.6 Integration with web application

The integration of YOLOv5 and YOLOv8 models with a web application will be achieved using Python Flask as the backend framework. This integration will enable seamless interaction between the trained object detection models and the web interface. Specifically, YOLOv5 will be utilized for detecting objects from uploaded images, allowing users to

submit images containing Mudras for real-time analysis. The YOLOv8 model will be employed for real-time detection, providing instant feedback to users through the web application's interface. Through the Flask framework, backend APIs will be developed to receive image uploads, process them using the YOLO models, and return the detected results to the frontend for display. This approach will leverage the efficiency of YOLO-based architectures for object detection while harnessing the flexibility and ease of deployment provided by Flask for web application integration.

5.2.7 User Interaction and Interface

The user interface and interaction for the Mudra Detection Tool web application are being developed using HTML, CSS, and JavaScript to create an intuitive and responsive user experience. The frontend interface includes a visually appealing dashboard where users can interact with the tool's features. HTML is used to structure the web pages, defining the layout and components such as buttons, forms, and image display areas. CSS is employed for styling and customization, ensuring a consistent and visually appealing design across different devices and screen sizes. JavaScript is instrumental in implementing dynamic behavior and interactivity, enabling functionalities such as uploading images, displaying detection results in real-time, and handling user inputs seamlessly. By leveraging these technologies, the web interface of the Mudra Detection Tool provides a user-friendly platform that empowers users to easily upload images, initiate gesture detection, and visualize the identified Mudras promptly.

5.3 User Interface Design

Our application's user interface has been meticulously designed to provide users with easy access to the model. By utilizing the capabilities of HTML, JavaScript, and CSS, users can easily navigate through our application. Our application starts with a home page, from which users can navigate to other pages, namely, the view mudra page, the learn page, and the feedback page.

Through the "Learn" page, users can upload the image of mudra based on which the Yolo V5 model will make predictions, and the name and image of the correct mudra will be displayed after a delay of 3 seconds. Also, there is a start camera button through which

the user can access the real-time detection and prediction feature of our application.

The next page is the "View Mudra" page. This page lists all the 25 mudras that we have selected for our project. When you click on a mudra name, the image and a short description of the mudra will be displayed.

Next, we have the "Feedback" page. Here, users can share their experience using our application along with any suggestions. Their name and email IDs will also be collected. These details are then stored as a CSV file.

5.4 Description of Implementation Strategies

The real time detection and prediction part of the application is implemented in the flask backend by setting the source of predict function of YOLO to webcam of the device. The predict function allows user to input data in different forms like individual image, video, etc. The function is present in the Ultralytics library. Ultralytics is a user-friendly AI platform with a no-code interface that supports deep learning frameworks for creating, training, and deploying machine learning models.

5.4.1 Implementation of YOLO

YOLO model was implemented in the following way:

- **Set Up Your Environment:** Install Python and PyTorch on your system. Set up any necessary dependencies like torchvision, NumPy, etc. Download the YOLOV8 and YOLOV5 model implementation (if not already available) or ensure you have the necessary code to train it.

- **Prepare Your Dataset:** Organize your dataset into the YOLO format, which typically involves creating annotations for each image with bounding boxes and class labels using Roboflow and collect its API Key. Create a data configuration file (usually in YAML format) that describes your dataset, including paths to images, class names, etc.

- **Set Up Your YOLO Model:** Load the YOLOV8 or YOLOV5 model architecture. This might involve importing a pre-defined model from a library like YOLOV5 or

```python
@app.route('/submit', methods=['POST'])
def submit_feedback():
    # Get the form data from the request
    name = request.form['name']
    email = request.form['email']
    rating = request.form['rating']
    message = request.form['message']
    # Create a dictionary to hold the feedback data
    feedback_data = {
        'Name': name,
        'Email': email,
        'Rating': rating,
        'Message': message
    }
    # Define the path to the CSV file
    csv_file_path = 'feedback_data.csv'
    # Write the feedback data to the CSV file
    with open(csv_file_path, mode='a', newline='') as csv_file:
        fieldnames = ['Name', 'Email', 'Rating', 'Message']
        writer = csv.DictWriter(csv_file, fieldnames=fieldnames)
        # Check if the CSV file is empty and write the header if needed
        if csv_file.tell() == 0:
            writer.writeheader()
        # Write the feedback data to the CSV file
        writer.writerow(feedback_data)
    return redirect(url_for('success'))
```

Figure 5.3: Flask Code for feedback submission.

```python
@app.route("/predict", methods=["POST"])
def predict():
    if 'file' not in request.files:
        return jsonify({'error': 'No file part'})
    file = request.files['file']
    if file.filename == '':
        return jsonify({'error': 'No selected file'})
    # Load the uploaded image
    image = Image.open(file)
    # Perform object detection using YOLO
    results = yolo.predict(image, save_txt=True)
    names=yolo.names
    # Save annotated images
    annotated_image_files = []
    for idx, result in enumerate(results):
        im_array = result.plot()  # Plot annotated image
        #predictedMudra=result.names()
        annotated_image = Image.fromarray(im_array[..., ::-1])  # Convert to PIL image
        # Save annotated image to BytesIO object
        image_buffer = BytesIO()
        annotated_image.save(image_buffer, format='JPEG')
        # Convert BytesIO object to base64 string
        annotated_image_base64 = base64.b64encode(image_buffer.getvalue()).decode('utf-8')
        # Save base64 string to list
        annotated_image_files.append(annotated_image_base64)
        # Save annotated image to file (optional)
        annotated_image.save(f'static/annotated_image_{idx}.jpg')
```

Figure 5.4: Flask Code for image upload using YOLO V5.

```
    for r in results:
        for c in r.boxes.cls:
            predicted_name=names[int(c)]
    # Prepare response data
    response_data = {
    'predicted_name': predicted_name,
    'annotated_images': annotated_image_files
}

    return jsonify(response_data)

@app.route('/realTime')
def realtime():
    try:
        model=YOLO('best_v8.pt')

        real_result=model.predict(source=0,show=True)

        return jsonify({"status": "success"})
    except Exception as e:
        # If an error occurs during prediction
        return jsonify({"status": "failure", "error": str(e)})
```

Figure 5.5: Flask Code for real-time prediction using YOLO V8 submission.

implementing the model architecture manually. Initialize the model with pre-trained weights if available or start training from scratch.

- **Configure Training Parameters:** Define your training parameters such as number of epochs (epochs), image size (imgsz), batch size, learning rate, etc. Set up data loaders to load batches of data during training.

- **Training Loop:** Iterate through your dataset for the specified number of epochs. Forward pass the images through the model to get predictions. Calculate the loss between predicted and ground truth bounding boxes and backpropagate to update the model weights. Monitor and log training metrics like loss, accuracy, etc.

- **Save Trained Model:** Save the trained model weights (yolov8n.pt for YOLO V8 and yolov5.pt for YOLO V5) for future inference or further training.

To evaluate the work done, metrices such as confusion matrix, precision-recall curves, precision-confidence curves, testing ,training and validation losses etc were used which is elaborated in Chapter 6.

The system implementation chapter delves into the development process and methodologies employed in creating the Interactive Dance Mudra Learning Tool. This chapter

outlines the key modules of the tool, including data preprocessing, segmentation, training, evaluation, mudra identification, and user interface design. The chapter also discusses the datasets identified, proposed methodologies/algorithms, and the integration of machine learning models with the web application interface. The datasets identified for the project consist of approximately 5000 images categorized into 25 classes of mudras. These images were preprocessed and segmented using Python, OpenCV, and TensorFlow, resulting in a curated dataset suitable for training machine learning models. The proposed methodology involves the use of YOLO and other algorithms for mudra detection and classification. The data collection and selection process involved gathering images from various sources and annotating them with bounding boxes using tools like Roboflow. Data augmentation techniques such as rotation and shear transformations were applied to enhance the dataset's diversity and robustness. The chapter discusses the selection of machine learning models, including CNN, VGG16, YOLOv5, and YOLOv8, with a focus on real-time performance and accuracy. The training process for these models, especially YOLOv5 and YOLOv8, involved fine-tuning and optimization to achieve high accuracy in mudra identification. Integration with the web application interface was achieved using Python Flask as the backend framework, enabling seamless interaction between the machine learning models and the frontend user interface. The frontend design, developed using HTML, CSS, and JavaScript, provides users with an intuitive platform to upload images, initiate mudra detection, and receive real-time predictions. Overall, the chapter provides a comprehensive overview of the system implementation, highlighting the methodologies, algorithms, datasets, and user interface design that collectively form the Interactive Dance Mudra Learning Tool.

Chapter 6

Results and Discussions

In this section, we present the results of our Bharatanatyam Dance Mudra Learning Tool developed using YOLOv5 and YOLOv8 object detection models. Our research focused on recognizing and classifying 25 distinct mudras commonly used in Bharatanatyam performances, with the goal of aiding dancers, instructors, and enthusiasts in learning and mastering these intricate hand gestures. Through rigorous testing and evaluation, we assessed the tool's performance, accuracy, and speed using a diverse dataset of Bharatanatyam mudras. Our results demonstrate the effectiveness and reliability of the tool in accurately identifying and categorizing mudras, highlighting the potential for digital technologies to enrich and democratize the learning of traditional art forms. The comparative analysis between YOLOv5 and YOLOv8 models provides valuable insights into their respective strengths and limitations in the context of Bharatanatyam mudra recognition, paving the way for future advancements and optimizations in dance education and technology integration. Our investigation into the results of the Dance Mudra Learning Tool involved extensive experimentation and validation to ensure robustness and practical applicability. The research employed a comprehensive methodology that encompassed data preprocessing, model training, and performance evaluation. The utilization of YOLOv5 and YOLOv8 models enabled us to achieve remarkable accuracy and efficiency in mudra detection and classification, with YOLOv8 showcasing notable improvements in certain scenarios such as low-light conditions and complex hand poses. The detailed analysis of detection metrics, including precision, recall, and F1 score, provided a nuanced understanding of model performance across different mudra classes and environmental variations. Additionally, user feedback and usability testing played a crucial role in refining the tool's interface, functionality, and user experience, ensuring seamless integration into dance pedagogy and practice.

6.1 Overview

YOLO V8m gives an accuracy of 85.3% for non augmented and no segmented data falling into 25 classes. YOLO V5 gives an accuracy of about 86% for non augmented and no segmented data falling into 25 classes. Mudras of various classes along with their meanings are displayed and also a description of each of the mudras . The name and accuracy of the static and real-time images are displayed along with its correction. Figures 6.1 and 6.2 show the results that was produced in YOLO V5. Figures 6.3 and 6.4 displays the real time results produced as result of training model in YOLO V8.

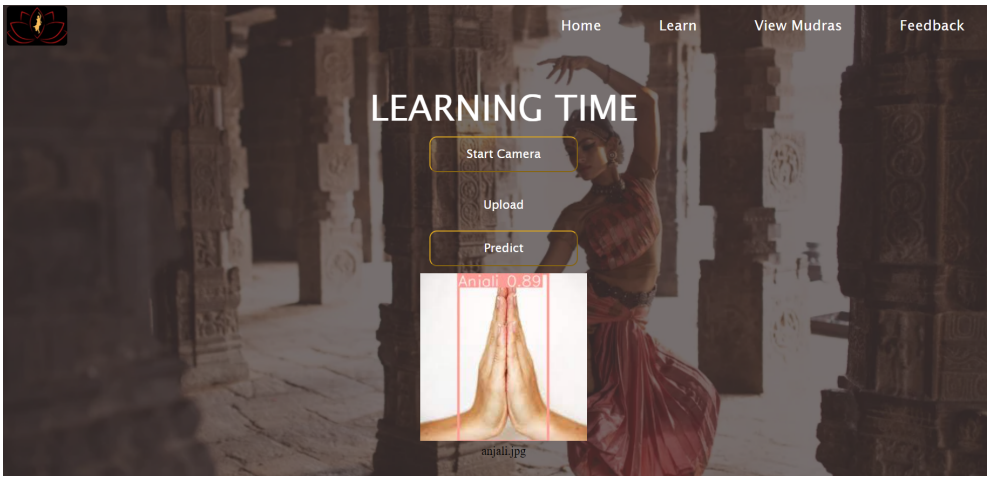


Figure 6.1: YOLO V5 Output.

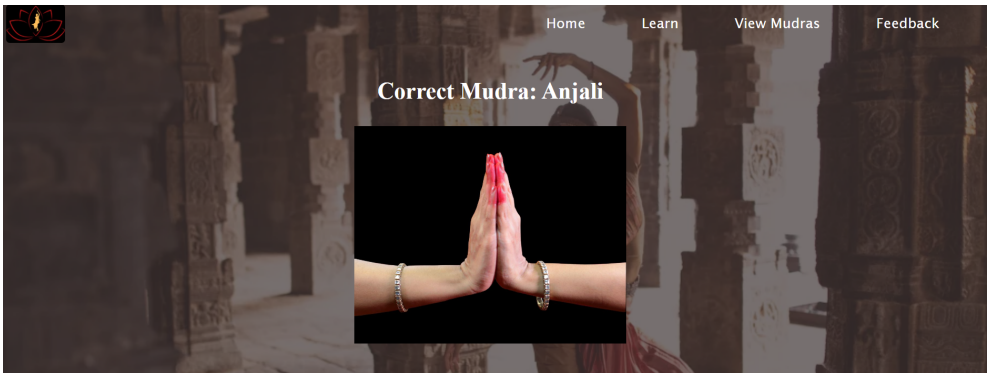


Figure 6.2: YOLO V5 Output.

Figure 6.3: YOLO V8 Output.

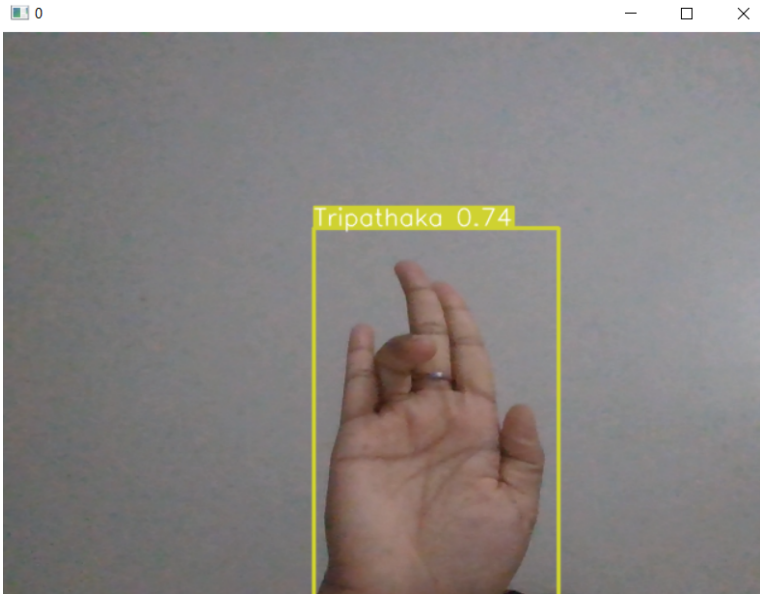


Figure 6.4: YOLO V8 Output.

6.1.1 Testing

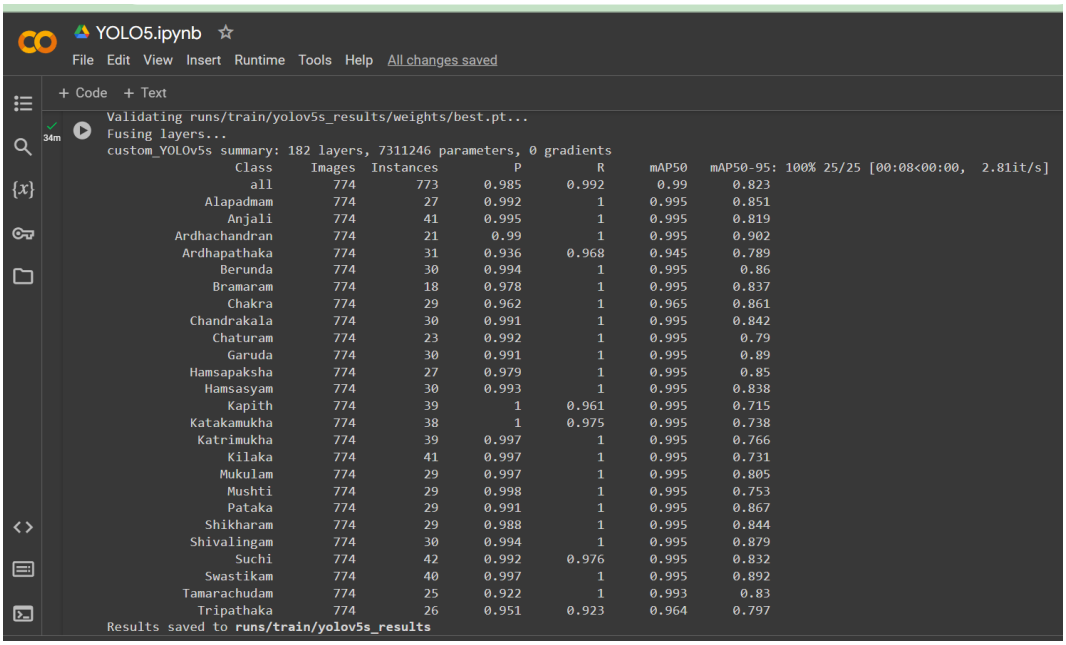


Figure 6.5: Training results of YOLOv5.

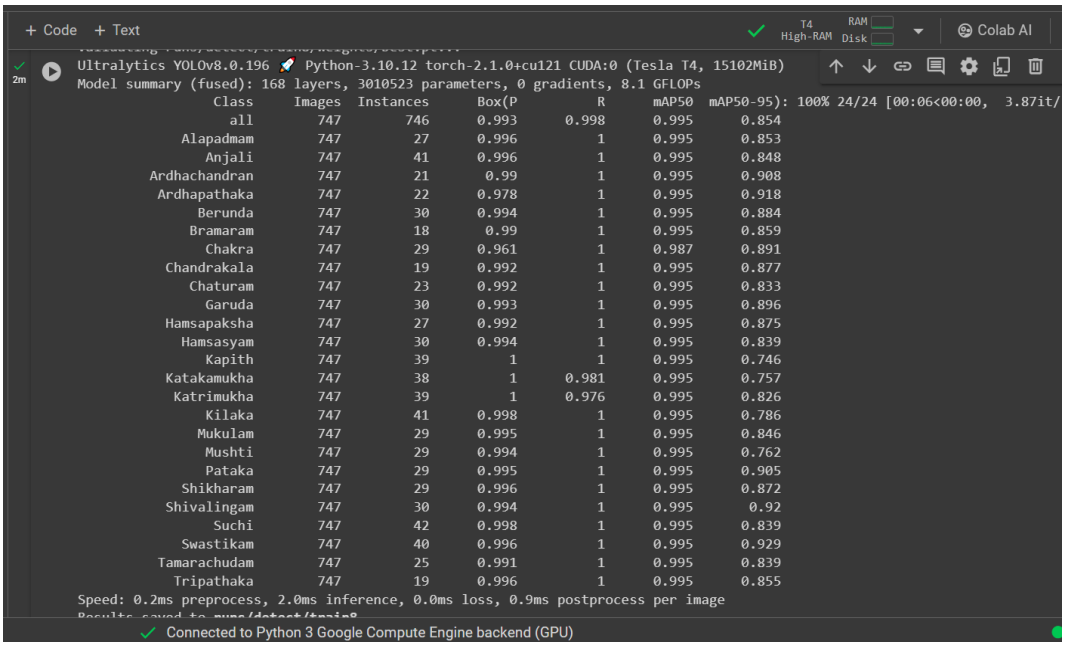


Figure 6.6: Testing Results of YOLOv8.

Figure 6.7: Home Page of Dance Mudra Learning Tool.

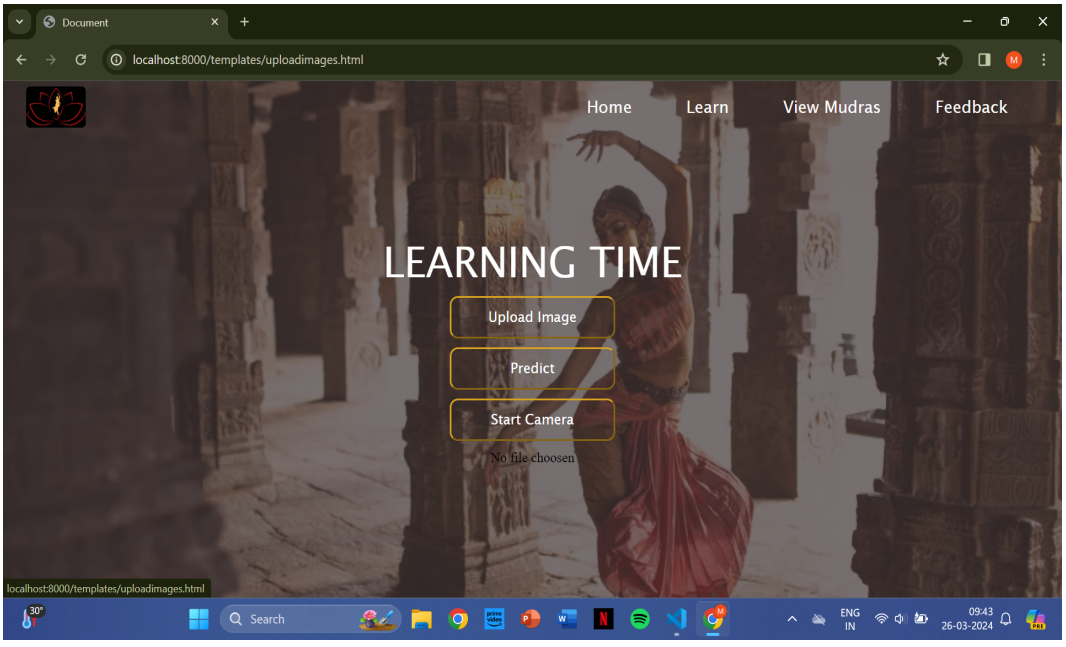


Figure 6.8: Learn Page of Dance Mudra Learning Tool.

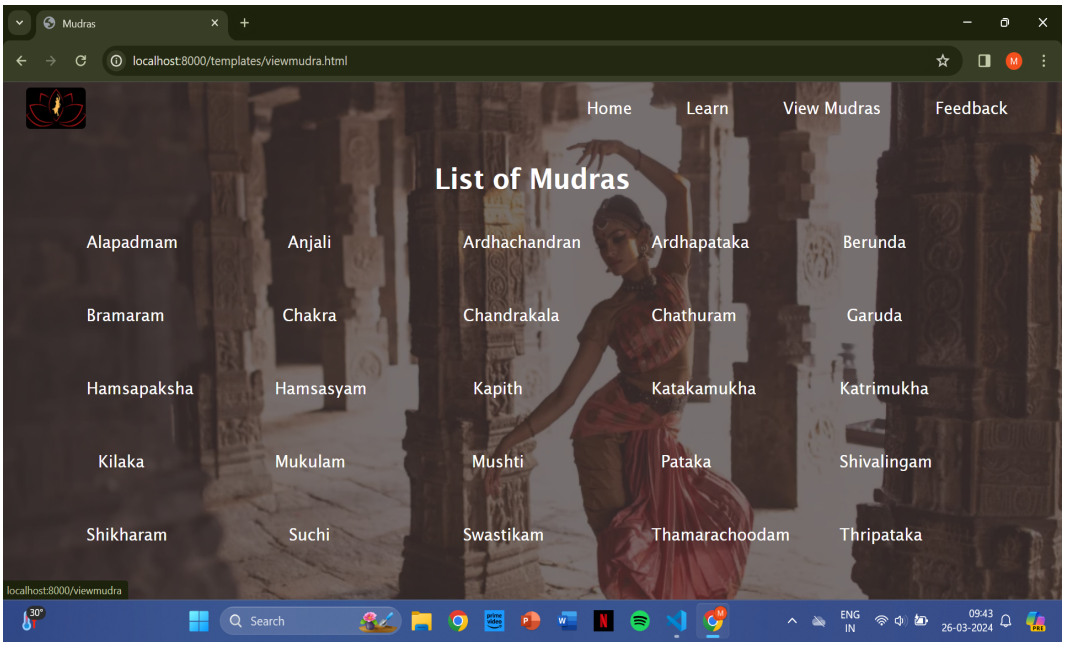


Figure 6.9: View Mudra Page of Dance Mudra Learning Tool.

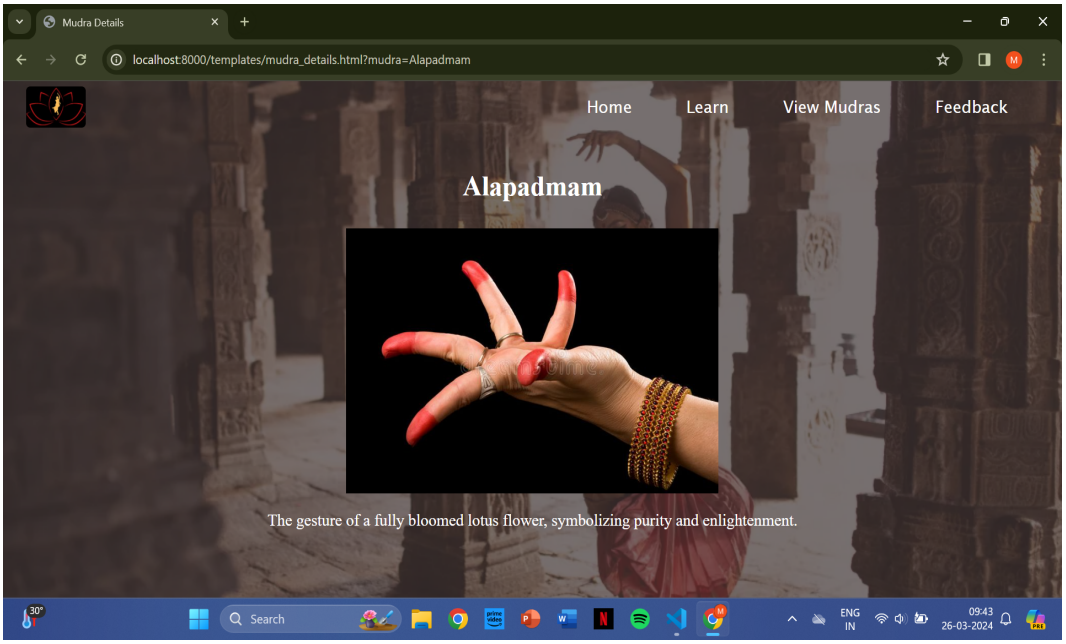


Figure 6.10: Alapadamam mudra displayed from view mudra page.

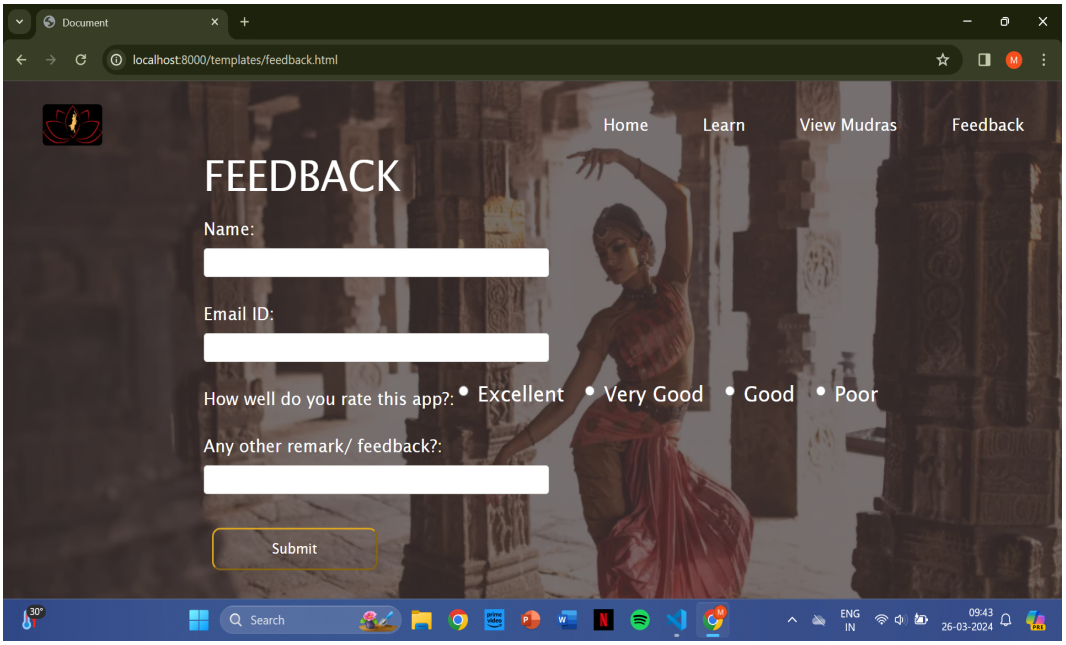


Figure 6.11: Feedback Page of Dance Mudra Learning Tool.

6.2 Quantitative Results

The confusion matrices created for YOLO V5 and V8 contains the normalized values of prediction which means that the rage is set as [0,1] and all values fall in between them. The value 1 shows that it predicts right always and 0 shows it never predicts it well. The confusion matrix for YOLO V5 is shown in Figure 6.12 and concluded the following:

- **Correct Classifications**: The model correctly classified images from several classes including Alapadmam, Anjali, Berunda, Bramaram (with one exception), Chakra, Chandrakala, Chaturam, Garuda, Hamsasyam, Pataka, Shikharam, and Swastikam.
- **Incorrect Classifications**: The model made some incorrect classifications including mistaking Kilaka for Mukulam, Tripathaka for background, and Tamarachudam for Suchi (and vice versa).
- **Uncertainties**: There are class labels with dashes (-) in the predicted column, indicating that the model is unsure about the classifications for those images.

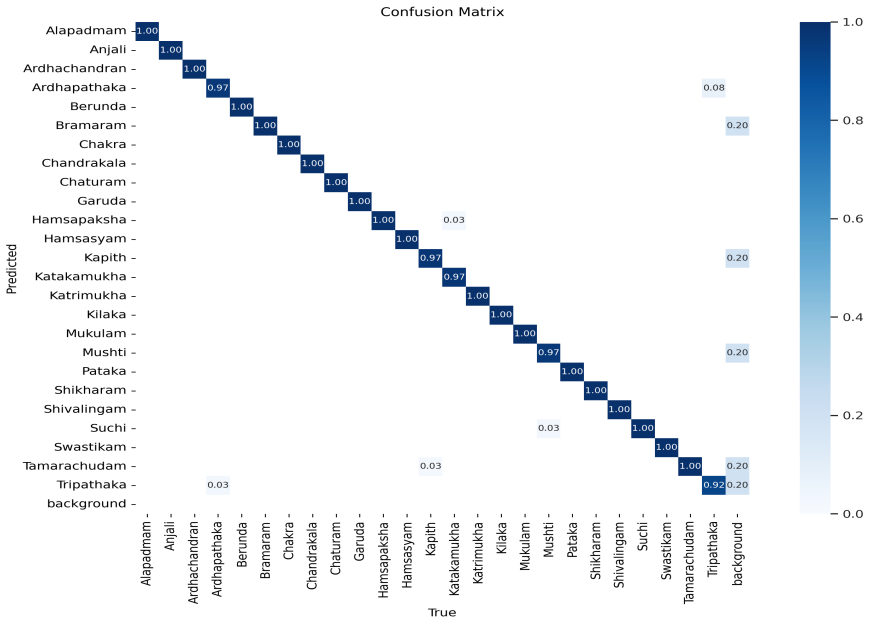


Figure 6.12: Normalized Confusion matrix for YOLO V5.

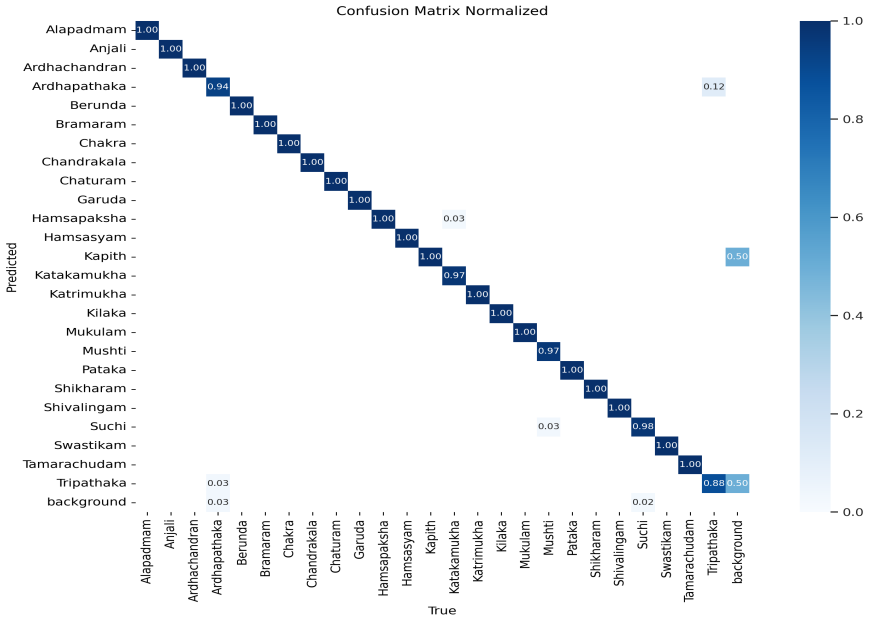


Figure 6.13: Normalized Confusion matrix for YOLO V8.

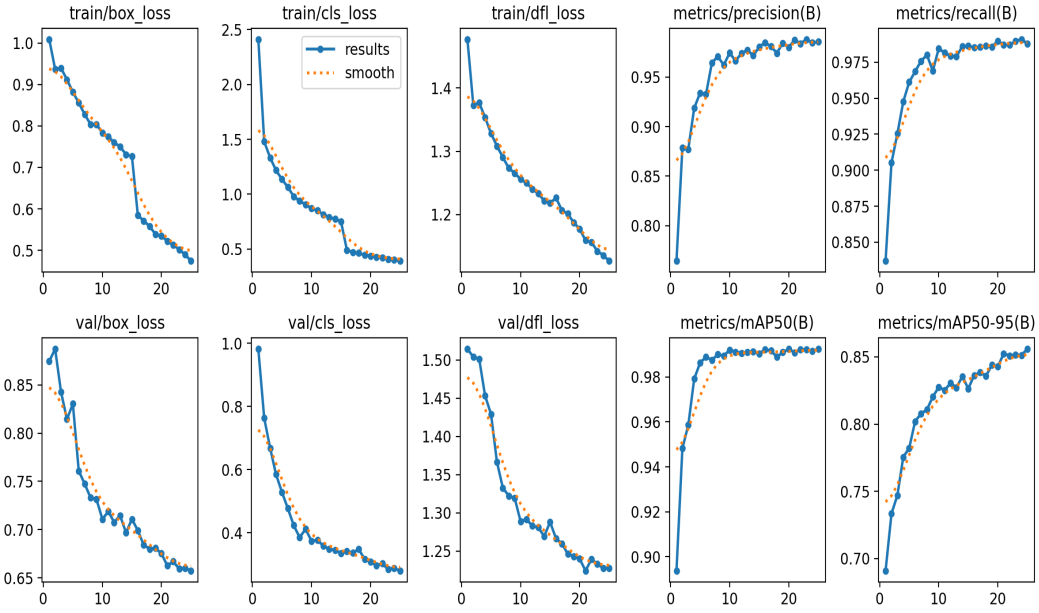


Figure 6.14: Training and validation losses for YOLO V8.

Here's a breakdown of some classifications and misclassifications in the matrix for YOLO V8 model as shown in Figure 6.13:

- **Correct Classifications**: The model correctly classified images from several classes including Alapadmam , Anjali , Berunda, Chakra , Chandrakala, Chaturam , Garuda , Hamsasyam, Pataka, Shikharam, and Swastikam .

- **Incorrect Classifications**: The model made some incorrect classifications including mistaking Kilaka for Mukulam , Tripathaka for background, and Tamarachudam for Suchi and vice versa (Suchi for Tamarachudam).

- **Uncertainties**: There are class labels with dashes (-) in the predicted column, indicating that the model is unsure about the classifications for those images (Ardhachandran, Bramaram, Hamsapaksha, Kapith, Katakamukha, Kilaka, Mukulam, Mushti, and Shivalingam).

  The image as shown in Figure 6.14 shows the training and validation losses for YOLO V8 model. It mentions mAP@0.5 values for each curve. mAP@0.5 refers to the mean Average Precision (mAP) at a threshold of 0.5. mAP is a common metric for summarizing precision-recall performance across various classification thresholds. A higher mAP@0.5 value generally indicates better overall performance. Figure 6.15

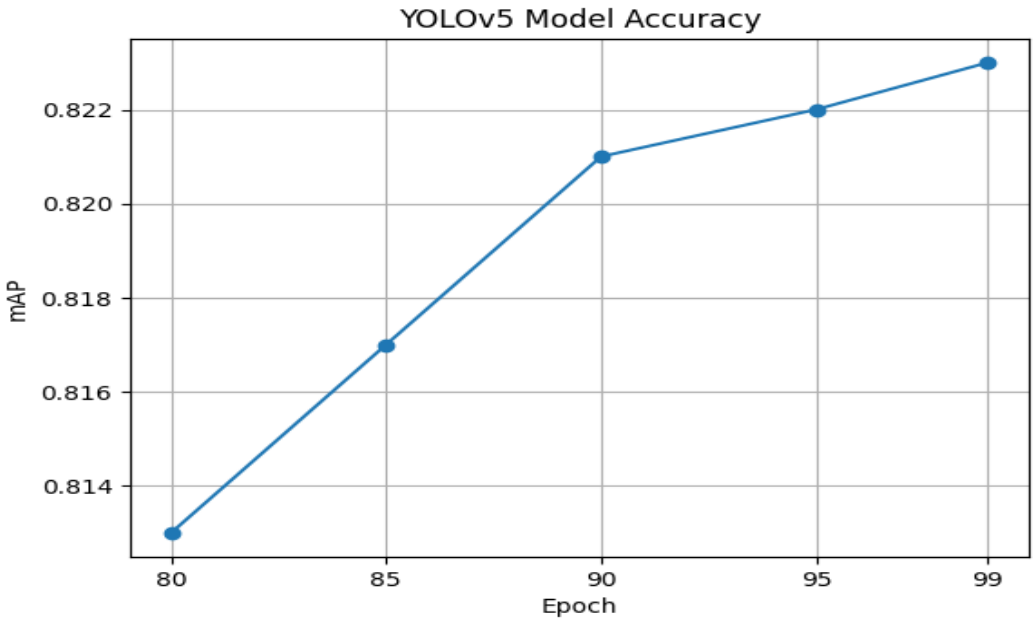


Figure 6.15: Accuracy graph for YOLO V5.

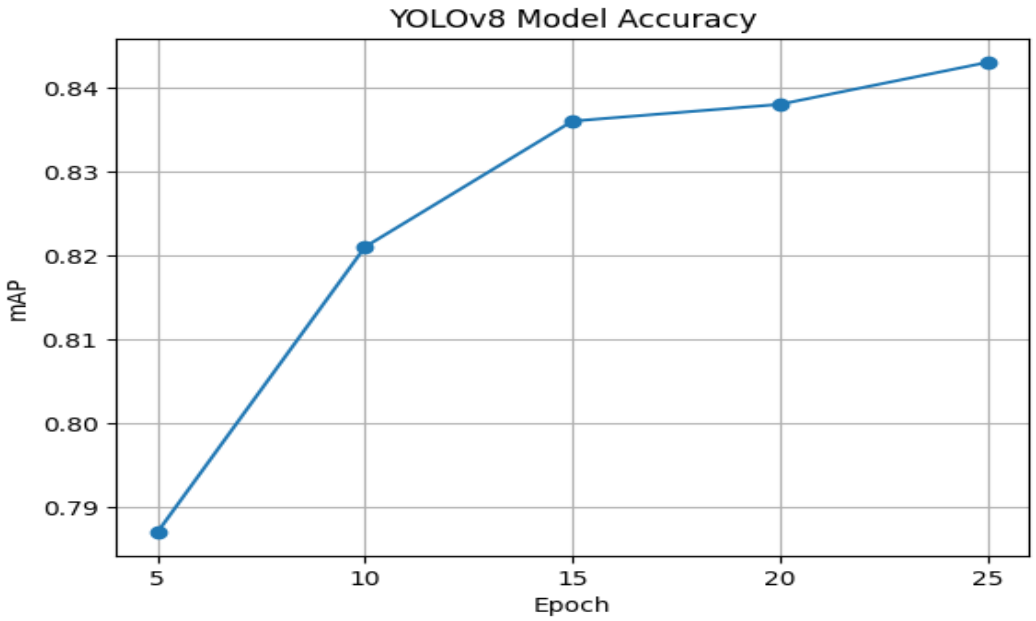


Figure 6.16: Accuracy graph for YOLO V5.

represents the Accuracy graph for YOLO V5 model. It shows how the values plotted between the mAP values and number of epochs the accuracy increases from 80% to 85% for epochs 80 to 100 respectively. Similarly, for YOLO V8 model the accuracy graph as shown in Figure 6.16, the accuracy increases from 75% to 86% for 5 to 20 epochs respectively. Increase in the number of epochs in both cases did not show any significant change in the accuracy of both the models.

Table 6.1: Comparison for accuracy and loss for various models.

MODEL	AUGMENTED/ NON AUGMENTED	SEGMENTED/ NON SEGMENTED	ACCURACY	LOSS
CNN	Non-Augmented	Segmented	45%	0.887
VGG16	Non-Augmented	Segmented	51%	0.883
Staged VGG16	Non- Augmented	Non Segmented	34%	1.8404
YOLOv8	Non- Augmented	Non Segmented	86%	0.22
YOLOv5	Augmented	Non Segmented	85%	0.0177

Table 6.1 shows the comparison for accuracy and loss for various models that we have used for training. The models were CNN, VGG16,Staged VGG16, YOLO V5 and YOLO V8 models. Out of which the CNN model and VGG16 models showed the least amount of accuarcy for the given dataset of mudras in 25 classes. VGG16 shows 91% only for 5 classes and reduced in accuracy further as number of classes increased.

Table 6.2 represents the comparison of various model parameters of the different models that we have used to train our model. It shows the training time taken to train the model in platforms such as Google Colab Pro and also local runtimes. The various batch sizes and number of epochs along with the accuracies are shown as a comparison in order to find out the best model.

Table 6.2: Comparison of various model parameters.

MODEL	TOTAL NUMBER OF IMAGES FOR TRAINING	BATCH SIZE	NUMBER OF EPOCHS	TRAINING TIME	ACCURACY
CNN	2000	32	50	1 hr	45%
VGG16	2000	16	10	1.5 hr	41%
Staged VGG16	28000	8	10	30 mins	34%
YOLOv8	5000	16	150	52 mins	86%
YOLOv5	5000	16	100	2.7 hrs	85%

6.3 Graphical Analysis

The Precision- Confidence curve for YOLO V5 and V8 are shown in Figures 6.17 and 6.18 respectively. A precision-confidence curve depicts the trade-off between precision and confidence in a model's predictions. It shows the follwoing:

- **X-axis (Confidence)**: This axis represents the confidence level of the model's predictions. Higher confidence indicates the model is more certain about its classification.
- **Y-axis (Precision)**: This axis represents the proportion of positive predictions that are actually correct. In other words, it reflects how accurate the model's predictions are among those it classifies with high confidence.

Interpreting the curve for YOLO V5 as shown in Figure 6.19, the curve starts at a perfect precision of 1.0 at a confidence of 0.965. This means that for the most confident predictions (confidence close to 0.965), all of them are precise (correct). As the confidence level decreases (moving left on the x-axis), the precision also decreases (moving down on the y-axis). This suggests that the model's predictions become less accurate as its confidence in those predictions weakens.

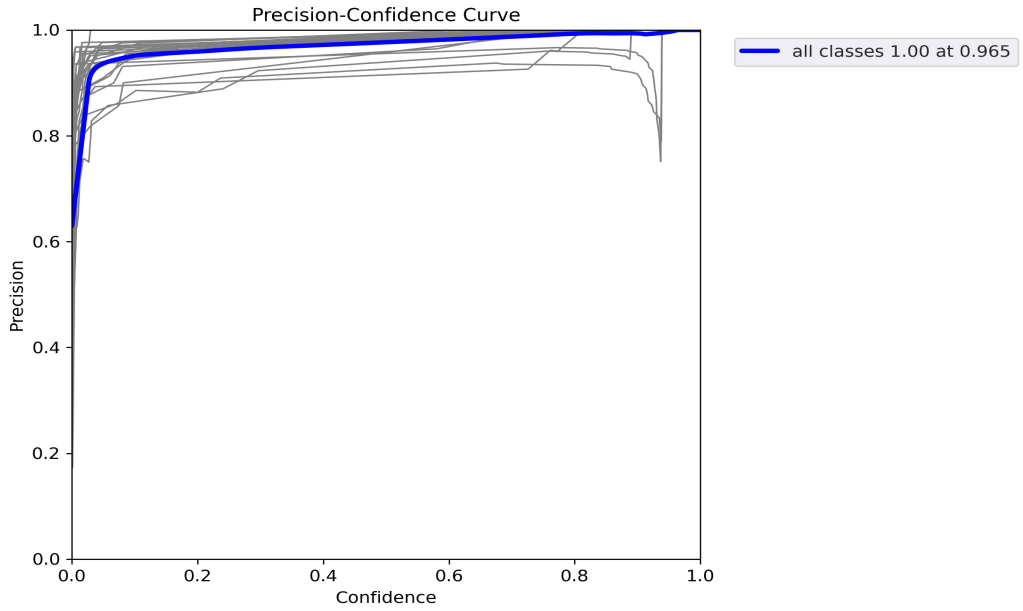


Figure 6.17: Precision-Confidence Curve for YOLO V5.

Interpreting the Precision-Confidence curve for YOLO V8 as in Figure 6.20, the curve starts at a perfect precision of 1.0 at a confidence of 0.827. This means that for the most confident predictions (confidence close to 0.827), all of them are precise (correct). The curve slopes downwards as confidence decreases. This means that the model's precision generally decreases as its confidence in the predictions weakens. For example, at a confidence of 0.2, the precision is around 0.4. This means that out of every 10 predictions the model makes with a confidence of 0.2, only about 4 are actually correct.

A precision-recall curve is a graph used to evaluate the performance of a binary classification model. It shows the trade-off between two important metrics: precision and recall. Precision is the fraction of positive predictions that are actually correct. In other words, it reflects how many of the images the model said were positive actually were positive. Recall is the fraction of positive cases that the model identified correctly. In simpler terms, it reflects how many of the actual positive images the model was able to identify.

- X-axis (Recall)

  This axis represents the recall of the model, which ranges from 0.0 (identified none of the positive cases) to 1.0 (identified all positive cases).

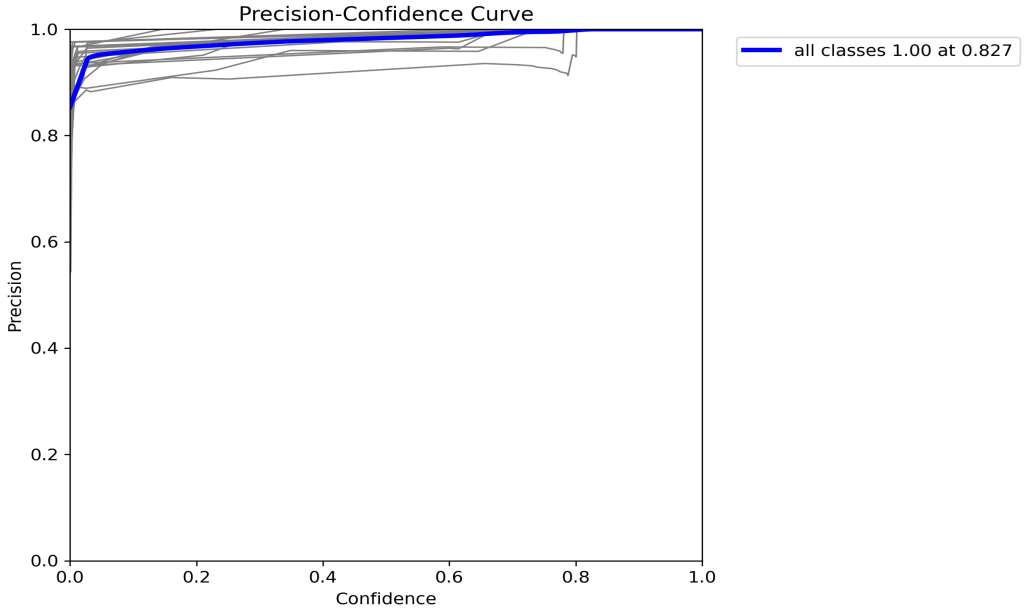


Figure 6.18: Precision-Confidence Curve for YOLO V8.

- Y-axis (Precision)

  This axis represents the precision of the model's predictions at a given recall level.

Interpreting the Curve as shown in Figure 6.16, the ideal scenario is for the curve to be as high as possible towards the top-left corner of the graph. This would indicate that the model has both high precision (most of the positive predictions are correct) and high recall (it identifies most of the positive cases). The curve starts at a precision of 1.0 and a recall of 0. This means that the model can perfectly identify a small number of positive cases with 100% precision. As the recall increases (the model identifies more positive cases), the precision starts to decrease. This is because the model starts including some negative cases in its predictions as well. The area under the curve is a popular metric for summarizing the overall performance of a model on a precision-recall curve. A higher area under the curve indicates a better overall performance.

The curve for YOLO V8 as shown in Figure 6.17, it starts with a precision of 1.0 and a recall of 0. This means that the model can perfectly identify a small number of positive cases with 100% precision.As the recall increases (the model identifies more positive cases), the precision starts to decrease. This is because the model starts including some negative cases in its predictions as well. The curve reaches a maximum precision of around 0.992 at a recall of 0.5. This means that the model can still achieve high precision (99% of positive

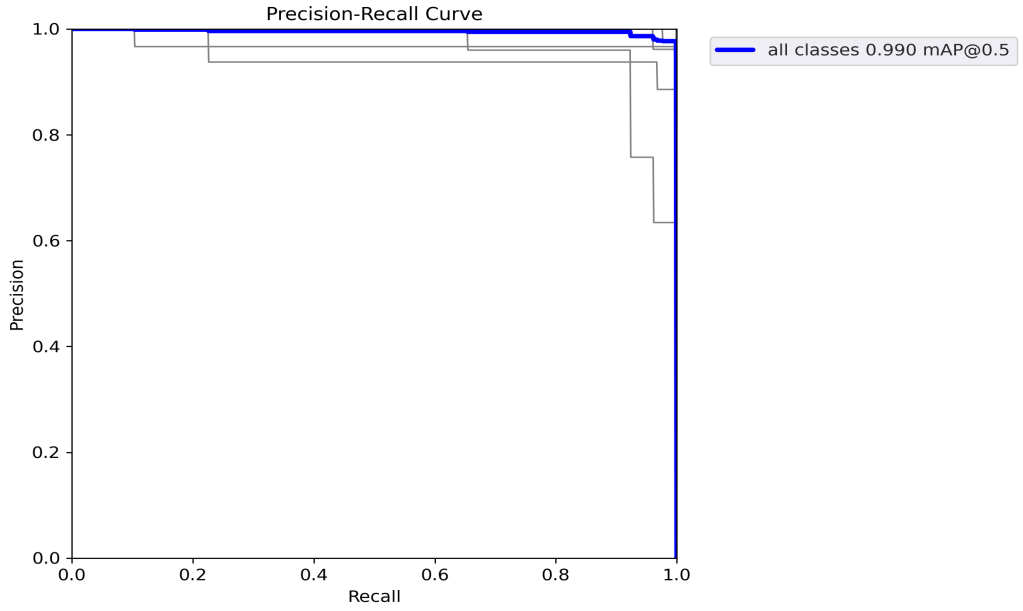


Figure 6.19: Precision-Recall Curve for YOLO V5

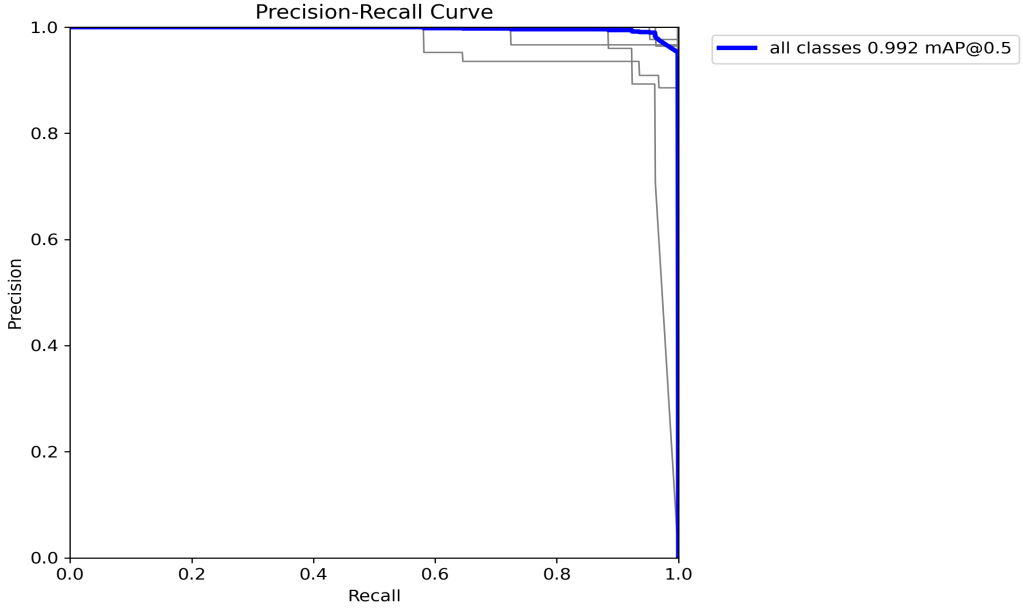


Figure 6.20: Precision-Recall Curve for YOLO V8

predictions are correct) when it identifies half of the positive cases. The text on the image mentions an mAP@0.5 value of 0.992, which likely refers to the mean Average Precision (mAP) at a threshold of 0.5. mAP is a common metric for summarizing precision-recall performance across various classification thresholds.

6.4 Discussion

The results obtained from the implementation of YOLO V5 and V8 models for the Bharatanatyam dance mudra learning tool provide valuable insights into their performance and suitability for this specific application. Firstly, both models achieved commendable accuracy rates, with YOLO V8 demonstrating a slightly higher accuracy of 86% compared to YOLO V5's 85.3%, indicating their effectiveness in accurately detecting and classifying mudras from the dataset comprising 25 classes. These results highlight the potential of object detection models like YOLO in facilitating the learning and understanding of intricate hand gestures and poses inherent in Bharatanatyam dance. The graphical analysis further elucidated the performance of the models, revealing precision-confidence curves and precision-recall curves that depict the trade-off between precision and confidence, as well as precision and recall, respectively. These curves offer valuable insights into the models' abilities to make accurate predictions across varying levels of confidence and recall, providing a nuanced understanding of their strengths and limitations. Moreover, the confusion matrices provided detailed information on the models' classification performance, highlighting instances of correct and incorrect classifications across different mudra classes. These insights are instrumental in identifying areas for improvement and fine-tuning the models to enhance their accuracy and robustness. Comparisons with other models, including CNN and VGG16, underscored the superior performance of YOLO V5 and V8 in terms of accuracy and efficiency, further validating their suitability for the proposed mudra learning tool. Overall, the results of this study demonstrate the efficacy of YOLO V5 and V8 models as powerful tools for mudra detection and classification in Bharatanatyam dance, paving the way for the development of innovative learning tools that leverage computer vision technology to enrich the study and practice of this ancient art form. Further research and experimentation could focus on refining the models and exploring additional techniques to enhance their performance and versatility in

real-world applications.

In this chapter, we explored the implementation and evaluation of YOLO V5 and V8 models within the context of a Bharatanatyam dance mudra learning tool for 25 mudra classes. The chapter encompassed a range of analyses, including quantitative results, graphical analyses, and comparisons with other models, providing a comprehensive overview of the models' performance and their implications for the proposed learning tool. The graphical analyses, including precision-confidence curves and precision-recall curves, offered deeper insights into the models' performance across varying levels of confidence, recall, and precision. These analyses highlighted the trade-offs inherent in the models' predictions and provided a nuanced understanding of their strengths and limitations in real-world scenarios. Furthermore, the comparison with other models such as CNN and VGG16 demonstrated the superior performance of YOLO V5 and V8 in terms of accuracy and efficiency, further validating their suitability for the proposed mudra learning tool.

Chapter 7

Conclusions & Future Scope

In this research, we delved into the fascinating field of Bharatanatyam hand mudras with the aim of classifying and correcting them using machine learning techniques in Python. Intricate hand movements known as mudras play a central role in the expression and storytelling of the Indian classical dance of Bharatanatyam. Using the power of machine learning, we aim to improve the accuracy and authenticity of mudra performance through automated classification and correction. In the initial phase of our research, the hand mudras of Bharatanatyam were thoroughly analyzed, their historical significance, cultural context and nuances of application were studied. This fundamental understanding paved the way for the development of a comprehensive database containing various examples of correctly and incorrectly performed mudras. We used the potential of Python, a versatile programming language, to implement machine learning algorithms that can distinguish between different mudra forms and provide corrective feedback. The classification of our project showed promising results that demonstrated the effectiveness of machine learning models to recognize different hand gestures. We used techniques such as YOLO V5 and V8 to use pre-trained models, adapting them to the complexity of Bharatanatyam mudras. The models were trained on our carefully curated dataset, achieving high accuracy in hand gesture classification. In addition, the repair module of our project has greatly contributed to the improvement of mudra performance. By analyzing deviations from ideal mudra forms, machine learning algorithms were able to provide constructive feedback to the dancers, helping to correct hand movements. This not only facilitates the learning of the practitioners but also ensures adherence to the traditional and aesthetic standards set by Bharatanatyam. The successful integration of machine learning into the hand mudra world of Bharatanatyam opens up new possibilities for the fusion of technology and art. As we continue to refine and expand our models, potential applications extend beyond real-time feedback corrections during performances, to personalized training modules and

even the creation of innovative choreographies. Our work is a testament to the harmonious coexistence of tradition and technology and illustrates how modern tools can be used to preserve and enrich ancient art forms. In conclusion, this research not only contributes to the field of Bharatanatyam, but also shows the wider possibilities that arise from the application of machine learning to cultural practice. The intersection of technology and art is a dynamic space with limitless potential, and learning about the hand mudras of Bharatanatyam is a pioneering step in this excitting journey.

The future scope includes improving models to recognize subtle variations and understand mudra context in dance sequences is a potential area for development and also to include more number of mudras. To speed up the learning process, individual training modules could be created based on individual dancer profiles. Integration of multimodal data such as audio and video would provide a more comprehensive analysis of Bharatanatyam performances. The project could contribute to the preservation and education of traditional dance forms by providing knowledge of cultural and historical significance. Collaboration with dance institutions could lead to the integration of technology into dance curricula, providing instructors with valuable tools. Adapting the technique to recognize gestures in other forms of classical or modern dance would expand its applicability. Investigating the accessibility of traditional dance forms, especially for people with different physical abilities, is a future development direction. Integrating machine learning into traditional art forms requires constant collaboration with practitioners, educators and technicians. The implications go beyond classification and remediation, with implications for real-time performance enhancement, personalized learning, and broader applications in dance and art education.

References

1 Herbaz, N., El Idrissi, H. and Badri, A., 2022. A Moroccan Sign Language Recognition Algorithm Using a Convolution Neural Network. Journal of ICT Standardization, 10(3), pp.411-426.

2 Jyotishman Bora, Saine Dehingia, Abhijit Boruah, Anuraag Anuj Chetia, Dikhit-Gogoi, Real-time Assamese Sign Language Recognition using MediaPipe and Deep Learning, Procedia Computer Science,Volume 218,2023,Pages 1384-1393,ISSN 1877-0509, https://doi.org/10.1016/j.procs.2023.01.117.

3 Ahmad Zaki Shukor, Muhammad Fahmi Miskon, Muhammad Herman Jamaluddin, Fariz bin Ali@Ibrahim, Mohd Fareed Asyraf, Mohd Bazli bin Bahar,A New Data Glove Approach for Malaysian Sign Language Detection,Procedia,Computer Science,Volume 76,2015,Pages 60-67,ISSN 1877-0509, https://doi.org/10.1016/j.procs.2015.12.276.

4 M. Mohandes, M. Deriche and J. Liu, "Image-Based and Sensor-Based Approaches to Arabic Sign Language Recognition," in IEEE Transactions on Human-Machine Systems, vol. 44, no. 4, pp. 551-557, Aug. 2014, doi: 10.1109/THMS.2014.2318280.

5 J. Rekha, J. Bhattacharya and S. Majumder, "Shape, texture and local movement hand gesture features for Indian Sign Language recognition," 3rd International Conference on Trendz in Information Sciences & Computing (TISC2011), Chennai, India, 2011, pp. 30-35, doi: 10.1109/TISC.2011.6169079.

6 M. K. Bhuyan, M. K. Kar and D. R. Neog, "Hand pose identification from monocular image for sign language recognition," 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Lumpur, Malaysia, 2011, pp. 378-383, doi: 10.1109/ICSIPA.2011.6144163

7 Tang, Xianlun & Yan, Zhenfu & Peng, Jiangping & Hao, Bohui & Wang, Huiming &

Li, Jie. ,2021. Selective spatiotemporal features learning for dynamic gesture recognition. Expert Systems with Applications. 169. 114499. 10.1016/j.eswa.2020.114499.

8 Ji, S., Xu, W., Yang, M., Yu, K. (2012). 3D convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35, 221–231.

9 Simonyan, K., Zisserman, A. (2014a). Two-stream convolutional networks for action recognition in videos. In Advances in neural information processing systems (pp. 568-576).

10 Miao, Q., Li, Y., Ouyang, W., Ma, Z., Xu, X., Shi, W., Cao, X. (2017). Multimodal gesture recognition based on the resc3d network. In Proceedings of the IEEE International Conference on Computer Vision Workshops (pp. 3047–3055).

11 Heng Wang, Cordelia Schmid. Action Recognition with Improved Trajectories. ICCV - IEEE International Conference on Computer Vision, Dec 2013, Sydney, Australia. pp.3551-3558, ff10.1109/ICCV.2013.441ff. ffhal-00873267v2f

12 A. P. Parameshwaran, H. P. Desai, M. Weeks and R. Sunderraman, "Unravelling of Convolutional Neural Networks through Bharatanatyam Mudra Classification with Limited Data," 2020 10th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 2020, pp. 0342-0347.

13 K.V.V.Kumar, and P.V.V Kishore, "Indian Classical Dance Mudra Classification Using HOG Features and SVM Classifier," International Journal of Electrical Computer Engineering (2088-8708), 7(5), 2017.

14 S.Saha, L.Ghosh, A.Konar, and R.Janarthanan, "Fuzzy L membership function based hand gesture recognition for Bharatanatyam dance," In 2013 5th International Conference and Computational Intelligence and Communication Networks (pp. 331-335). IEEE. September 2013.

15 Anami, Basavaraj Bhandage, Venkatesh. (2018). A vertical-horizontal-intersections feature based method for identification of bharatanatyam double hand mudra images. Multimedia Tools and Applications. 77. 10.1007/s11042-018-6223-y.

16 B.S.Anami, and V.A.Bhandage, "A Comparative Study of Suitability of Certain Features in Classification of Bharatanatyam Mudra Images Using Artificial Neural Network," Neural Processing Letters, 50(1), pp.741-769, 2019.

17 Mujahid, A.; Awan, M.J.; Yasin, A.; Mohammed, M.A.; Damaševi˘cius, R.; Maskeliunas, R.; ¯ Abdulkareem, K.H. Real-Time Hand Gesture Recognition Based on Deep Learning YOLOv3 Model. Appl. Sci. 2021, 11, 4164. https://doi.org/10.3390/app11094

18 Ashiquzzaman, A.; Lee, H.; Kim, K.; Kim, H.-Y.; Park, J.; Kim, J. Compact Spatial Pyramid Pooling Deep Convolutional Neural Network Based Hand Gestures Decoder. Appl. Sci. 2020, 10, 7898.

19 Tran, D.; Ho, N.; Yang, H.; Baek, E.; Kim, S.; Lee, G. Real-time hand gesture spotting and recognition using RGB-D camera and 3D convolutional neural network. Appl. Sci. 2020, 10, 722.

20 Qassim, H.; Verma, A.; Feinzimer, D. Compressed residual-VGG16 CNN model for big data places image recognition. In Proceedings of the 2018 IEEE 8th Annual Computing and Communication Workshop and Conference (CCWC), Las Vegas, NV, USA, 8–10 January 2018; pp. 169–175.

1. Xu, Renjie Lin, Haifeng Lu, Kangjie Cao, Lin Liu, Yunfei. (2021). A Forest Fire Detection System Based on Ensemble Learning. Forests. 12. 217. 10.3390/f12020217.

2. Bai, Ruihan Shen, Feng Wang, Mingkang Lu, Jiahui Zhang, Zhiping. (2023). Improving Detection Capabilities of YOLOv8-n for Small Objects in Remote Sensing Imagery: Towards Better Precision with Simplified Model Complexity. 10.21203/rs.3.rs-3085871/v1.

Appendix A: Presentation

100% OUTPUT PRESENTATION : INTERACTIVE DANCE MUDRA LEARNING TOOL

Presented by
Nekha S Thomas-U2003149 Sandra Sunil-U2003186
Pooja Menon-U2003160 Sayujya Salim-U2003189

Guide
Ms.Amitha Mathew
Department of CSE,RSET

May 15, 2024

Problem Definition

Develop a web application that uses deep learning techniques to recognize and provide feedback on the correctness of Bharatanatyam mudras (hand gestures) based on uploaded images.

Project Objectives

MUDRA RECOGNITION
To develop a system that can accurately recognize and identify Bharatanatyam mudras from images.

USER-FRIENDLY INTERFACE
Create a user-friendly web application where users can easily interact with the model.

CORRECTION AND FEEDBACK
Once a mudra is recognized, the system should provide feedback and corrections to the user

Novelty of Idea and Scope of Implementation

- To assist Bharatanatyam learners and enthusiasts in mastering the art of mudra execution, merging tradition with technology.
- To Safeguard the Rich Tradition of Bharatanatyam in an Accessible Manner.
- To Offer Learners a Digital Platform with Expert Insights.

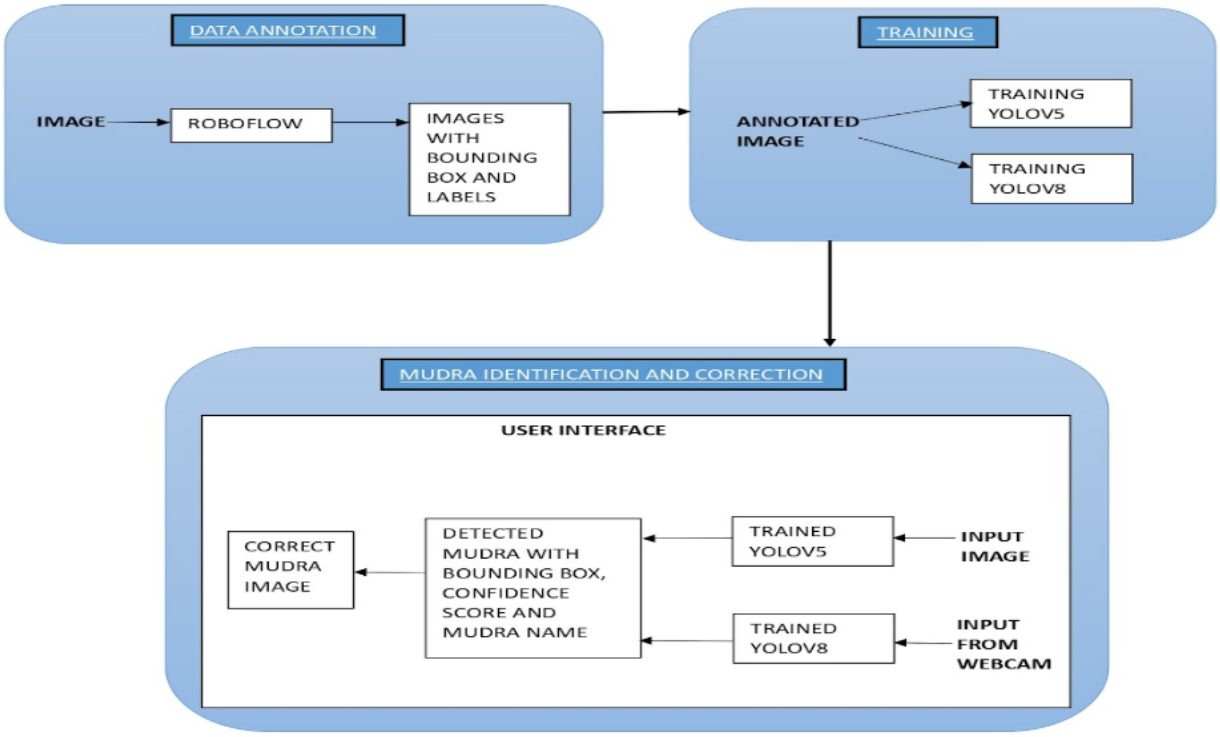
Literature Review

Method	Advantage	Disadvantage
A Moroccan Sign Language Recognition Algorithm Using a Convolution Neural Network [1]	One advantage of the proposed method is its high accuracy rate of 98.7%. This is a significant improvement compared to other state-of-the-art models.	One disadvantage of this method is that it requires a large dataset for training the model . This can be time-consuming and resource-intensive, especially if the dataset needs to be manually labeled.
Sign Language Recognition using Mediapipe and Deep Learning [2]	MediaPipe offers a user-friendly API and pre-trained models, making it accessible for developers with varying levels of expertise. The modular design simplifies the integration of features into applications.	It may not be as customizable as building a solution from scratch, especially for highly specialized or domain-specific requirements.
Hand Gesture Recognition using YOLOv3 Model [17]	YOLOv3 has real-time processing capabilities, making it well suited for applications such as video surveillance, and its ability to handle different types of objects with high accuracy.	It has limitations in accurate localization, struggling with small objects, and its dependence on a large and diverse training dataset.

Literature Review

Unravelling of Convolutional Neural Networks through Bharatanatyam Mudra Classification with Limited Data [12]	Its ability to automatically learn features from the data without manual intervention.	The potential requirement for a large amount of data for effective leveraging of machine learning techniques.
Selective Spatiotemporal Features Learning For Dynamic Gesture Recognition [7]	Adapts ResC3D and ConvLSTM with a dynamic select mechanism for simultaneous short-term and long-term spatiotemporal feature learning.	Highly complex and time consuming.

Methodology



Methodology

DATA ANNOTATION

The 5000 images are manually labeled by drawing bounding boxes using Roboflow.

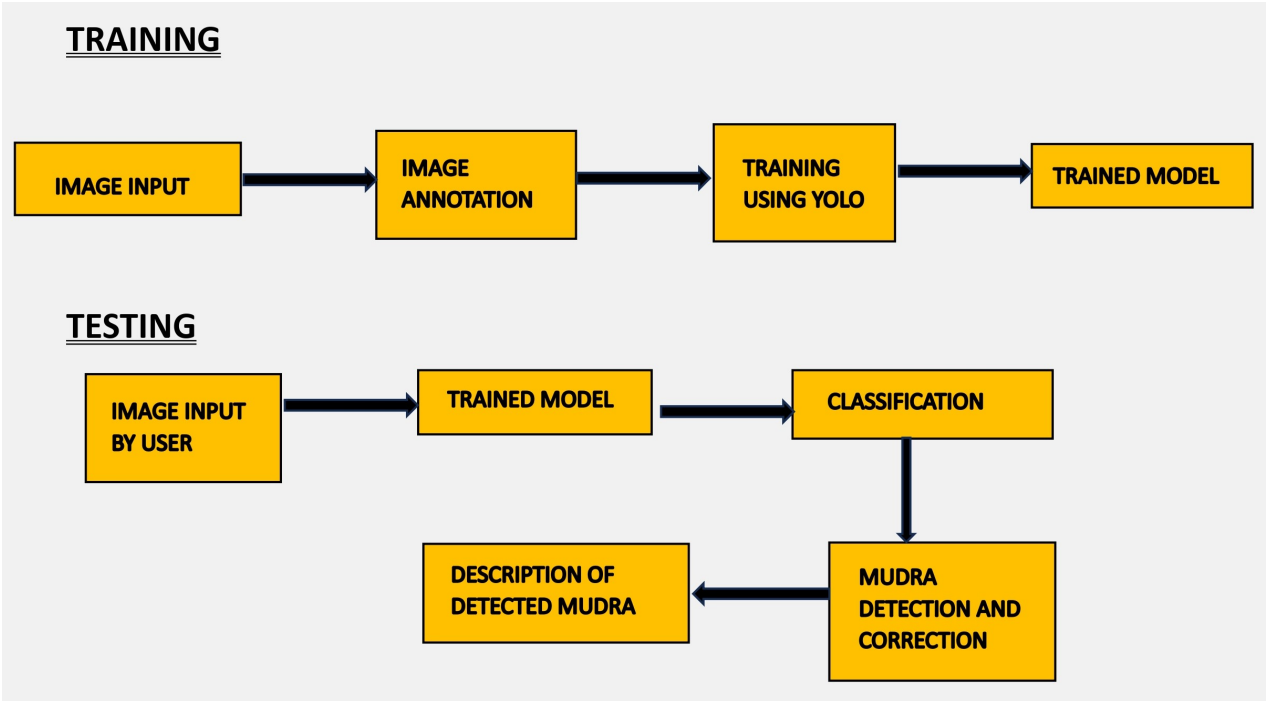
TRAINING

Annotated images are trained using YOLOv5 and YOLOv8.

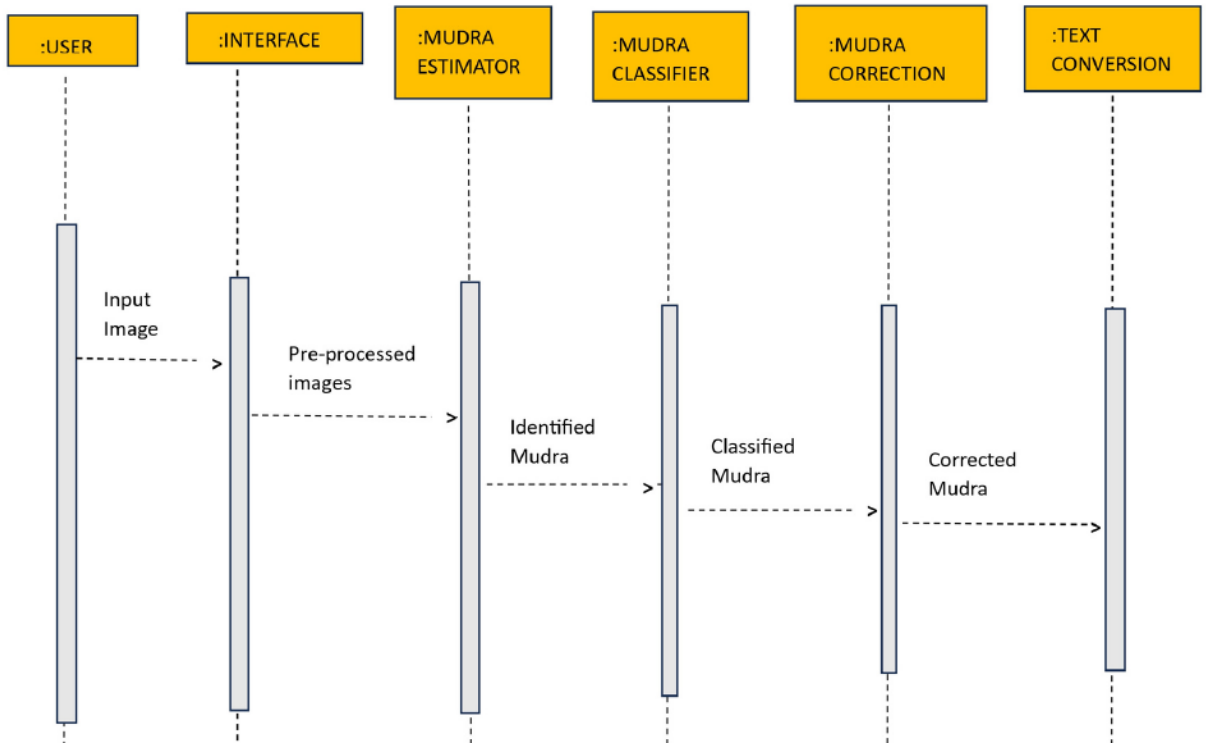
IDENTIFICATION AND CORRECTION

The image is given as input to the trained model, and the output includes the detected mudra with a bounding box, confidence score, and mudra name, which is then directed to the correct mudra image.

Architecture Diagram



Sequence Diagram



YOLOv5 Results

YOLO5.ipynb

File Edit View Insert Runtime Tools Help All changes saved

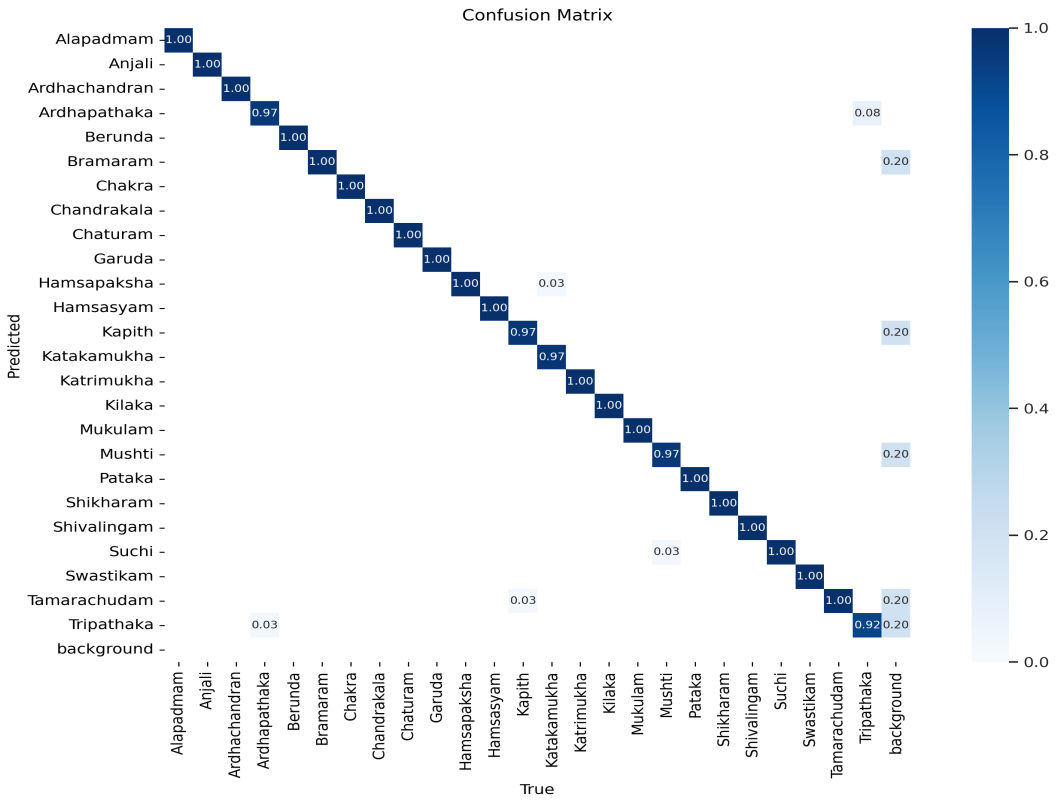
+ Code + Text

```
Validating runs/train/yolov5s_results/weights/best.pt...
Fusing layers...
custom_YOLOv5s summary: 182 layers, 7311246 parameters, 0 gradients
```

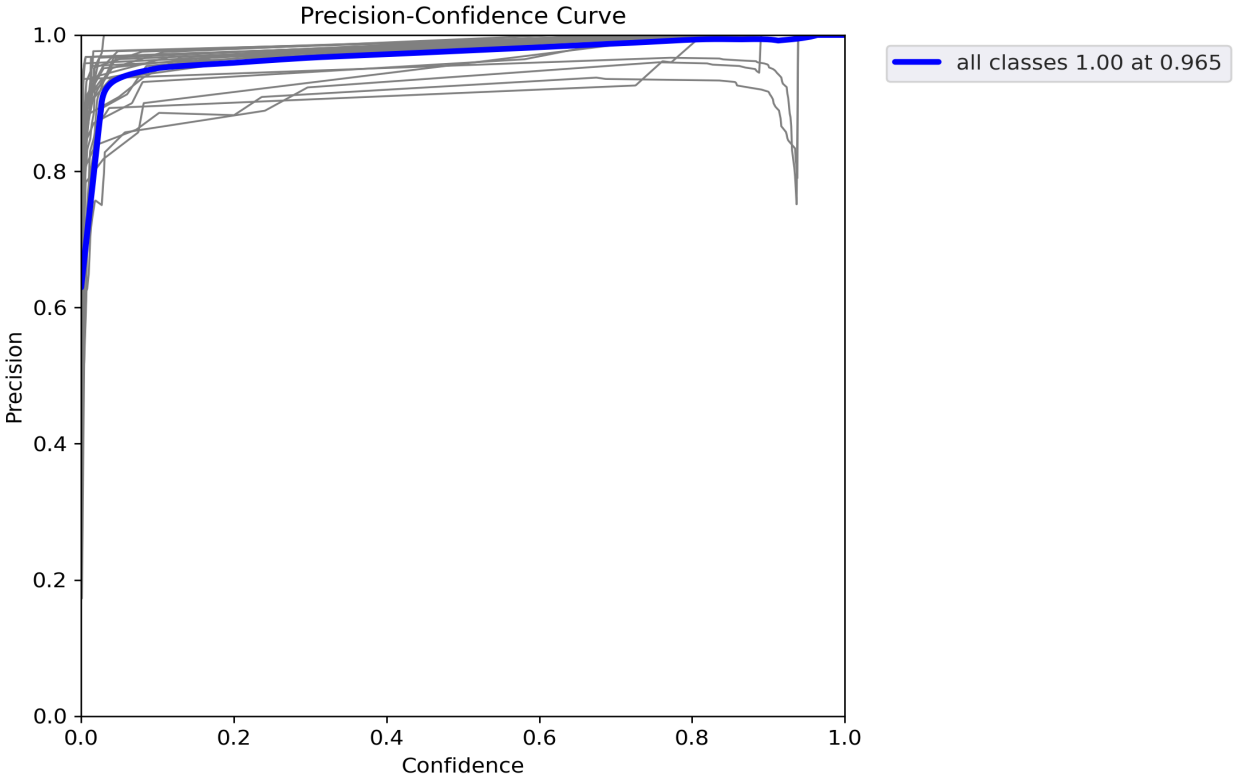
Class	Images	Instances	P	R	mAP50	mAP50-95: 100% 25/25 [00:08<00:00, 2.81it/s]
all	774	773	0.985	0.992	0.99	0.823
Alapadmam	774	27	0.992	1	0.995	0.851
Anjali	774	41	0.995	1	0.995	0.819
Ardhachandran	774	21	0.99	1	0.995	0.902
Ardhapathaka	774	31	0.936	0.968	0.945	0.789
Berunda	774	30	0.994	1	0.995	0.86
Bramaram	774	18	0.978	1	0.995	0.837
Chakra	774	29	0.962	1	0.965	0.861
Chandrakala	774	30	0.991	1	0.995	0.842
Chaturam	774	23	0.992	1	0.995	0.79
Garuda	774	30	0.991	1	0.995	0.89
Hamsapaksha	774	27	0.979	1	0.995	0.85
Hamsasyam	774	30	0.993	1	0.995	0.838
Kapith	774	39	1	0.961	0.995	0.715
Katakamukha	774	38	1	0.975	0.995	0.738
Katrimukha	774	39	0.997	1	0.995	0.766
Kilaka	774	41	0.997	1	0.995	0.731
Mukulam	774	29	0.997	1	0.995	0.805
Mushti	774	29	0.998	1	0.995	0.753
Pataka	774	29	0.991	1	0.995	0.867
Shikharam	774	29	0.988	1	0.995	0.844
Shivalingam	774	30	0.994	1	0.995	0.879
Suchi	774	42	0.992	0.976	0.995	0.832
Swastikam	774	40	0.997	1	0.995	0.892
Tamarachudam	774	25	0.922	1	0.993	0.83
Tripathaka	774	26	0.951	0.923	0.964	0.797

```
Results saved to runs/train/yolov5s_results
```

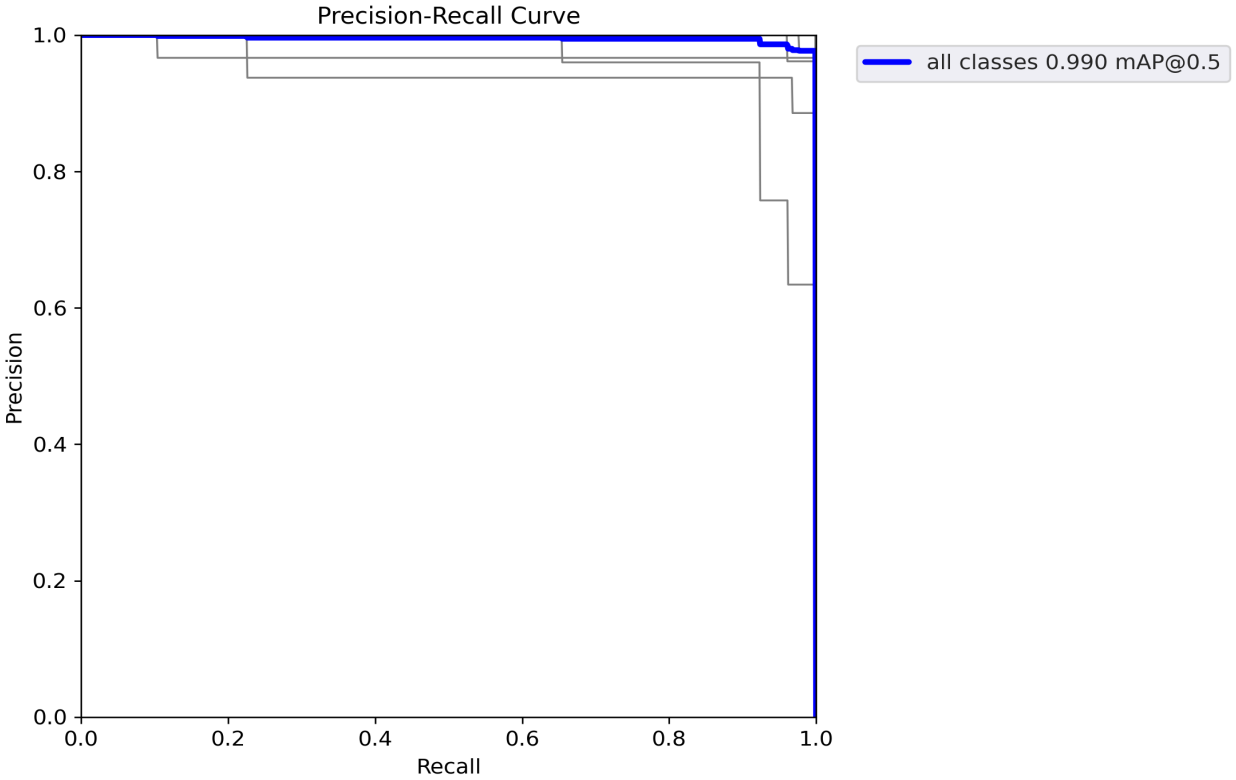

YOLOv5 Results



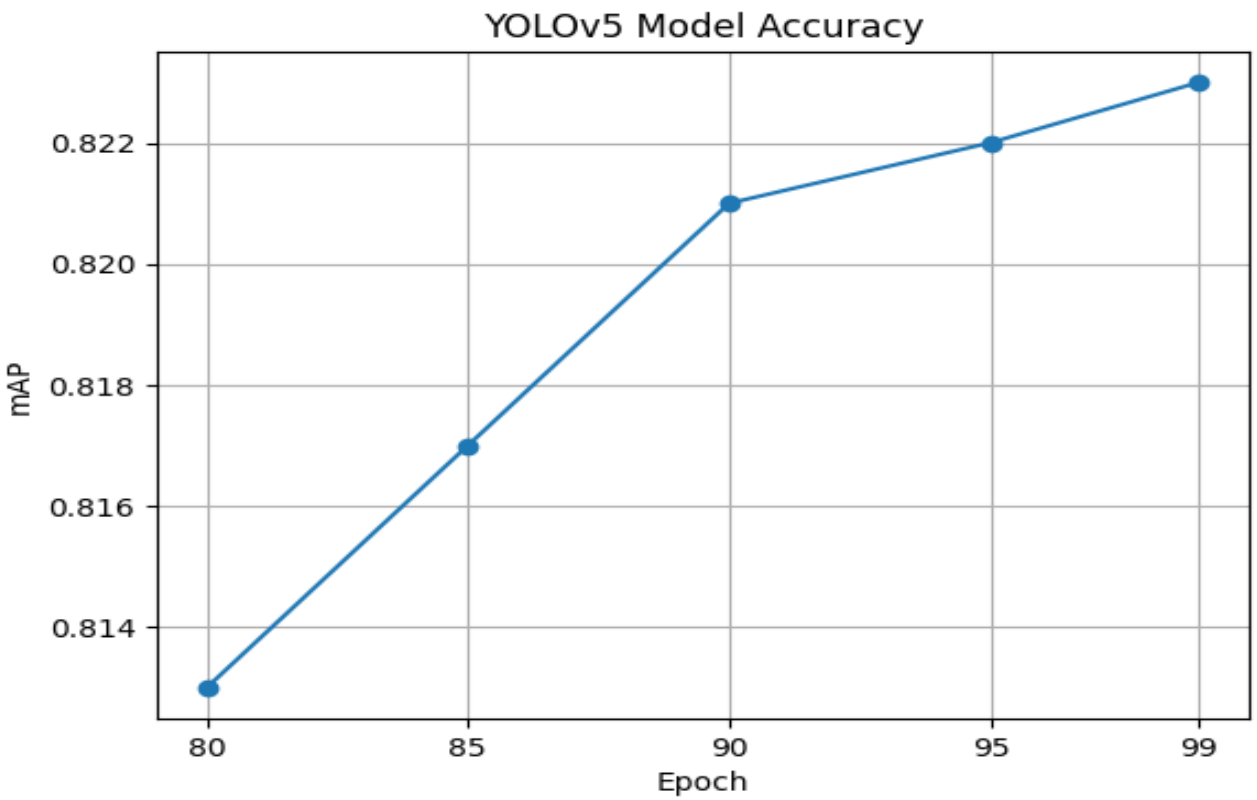
YOLOv5 Results



YOLOv5 Results



YOLOv5 Results



YOLOv8 Results

+ Code + Text

T4 High-RAM RAM Disk Colab AI

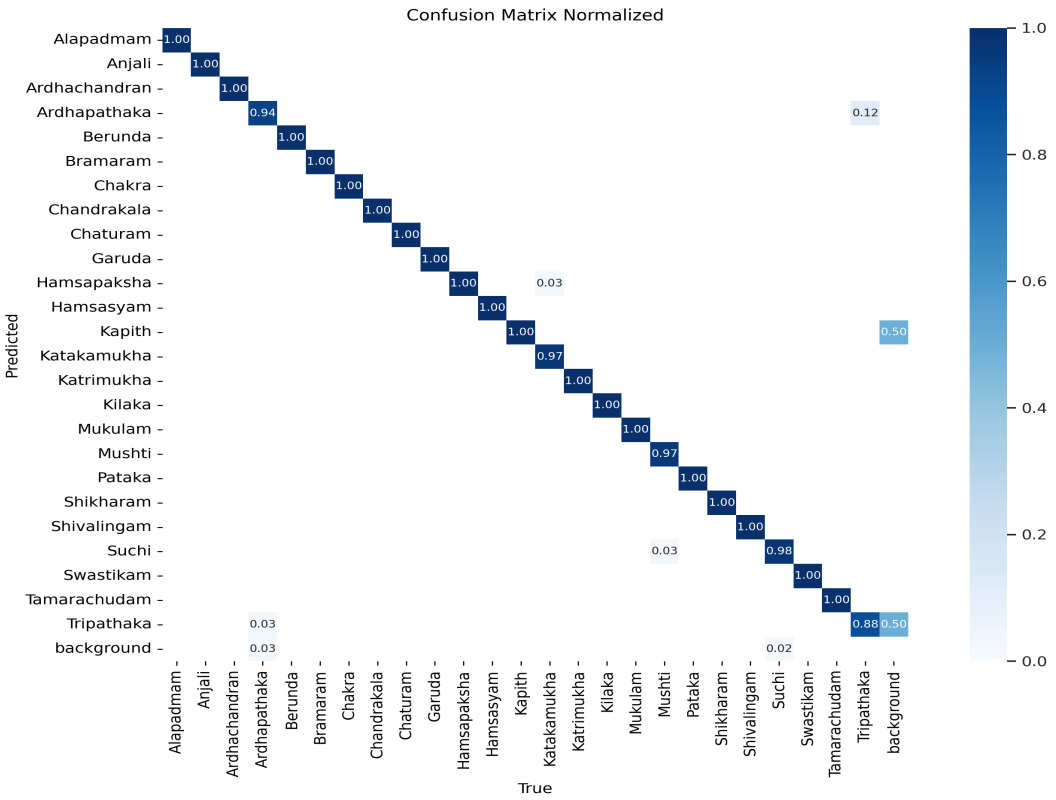
Ultralytics YOLOv8.0.196 🚀 Python-3.10.12 torch-2.1.0+cu121 CUDA:0 (Tesla T4, 15102MiB)
Model summary (fused): 168 layers, 3010523 parameters, 0 gradients, 8.1 GFLOPs

Class	Images	Instances	Box(P	R	mAP50	mAP50-95): 100% 24/24 [00:06<00:00, 3.87it/
all	747	746	0.993	0.998	0.995	0.854
Alapadmam	747	27	0.996	1	0.995	0.853
Anjali	747	41	0.996	1	0.995	0.848
Ardhachandran	747	21	0.99	1	0.995	0.908
Ardhapathaka	747	22	0.978	1	0.995	0.918
Berunda	747	30	0.994	1	0.995	0.884
Bramaram	747	18	0.99	1	0.995	0.859
Chakra	747	29	0.961	1	0.987	0.891
Chandrakala	747	19	0.992	1	0.995	0.877
Chaturam	747	23	0.992	1	0.995	0.833
Garuda	747	30	0.993	1	0.995	0.896
Hamsapaksha	747	27	0.992	1	0.995	0.875
Hamsasyam	747	30	0.994	1	0.995	0.839
Kapith	747	39	1	1	0.995	0.746
Katakamukha	747	38	1	0.981	0.995	0.757
Katrimukha	747	39	1	0.976	0.995	0.826
Kilaka	747	41	0.998	1	0.995	0.786
Mukulam	747	29	0.995	1	0.995	0.846
Mushti	747	29	0.994	1	0.995	0.762
Pataka	747	29	0.995	1	0.995	0.905
Shikharam	747	29	0.996	1	0.995	0.872
Shivalingam	747	30	0.994	1	0.995	0.92
Suchi	747	42	0.998	1	0.995	0.839
Swastikam	747	40	0.996	1	0.995	0.929
Tamarachudam	747	25	0.991	1	0.995	0.839
Tripathaka	747	19	0.996	1	0.995	0.855

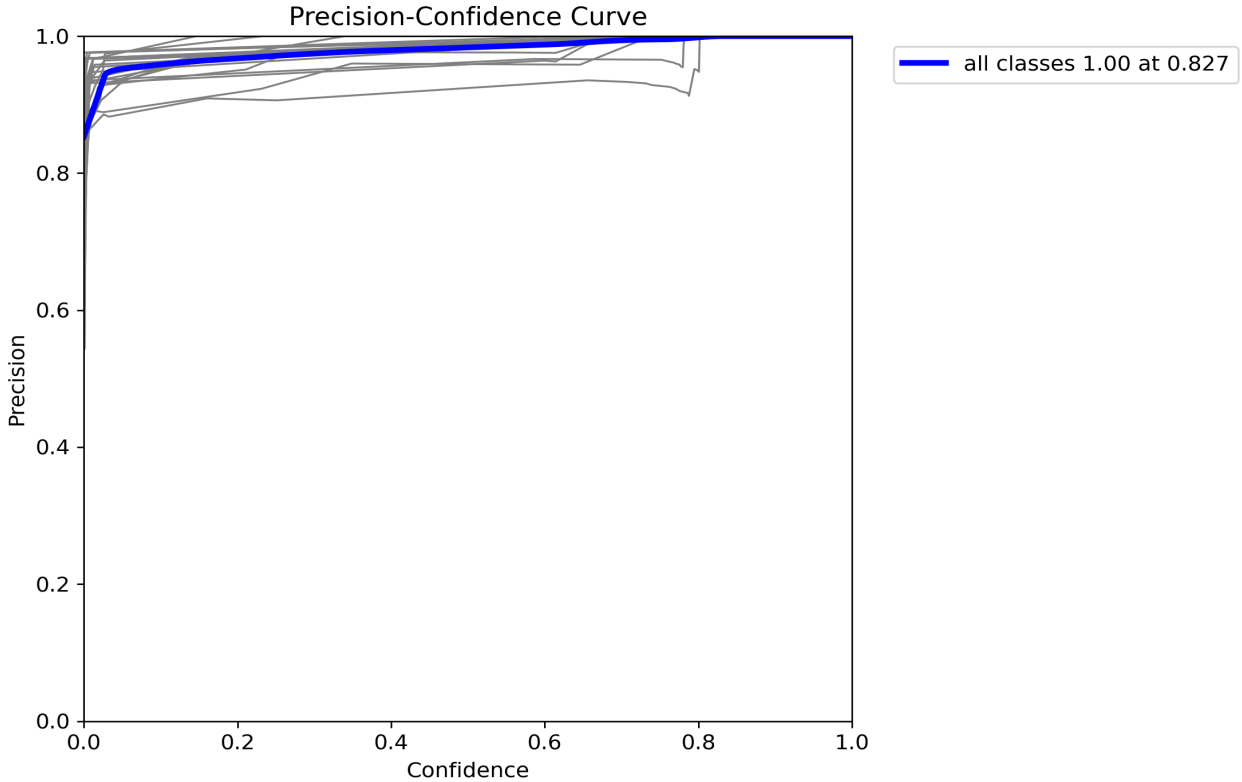
Speed: 0.2ms preprocess, 2.0ms inference, 0.0ms loss, 0.9ms postprocess per image

Connected to Python 3 Google Compute Engine backend (GPU)

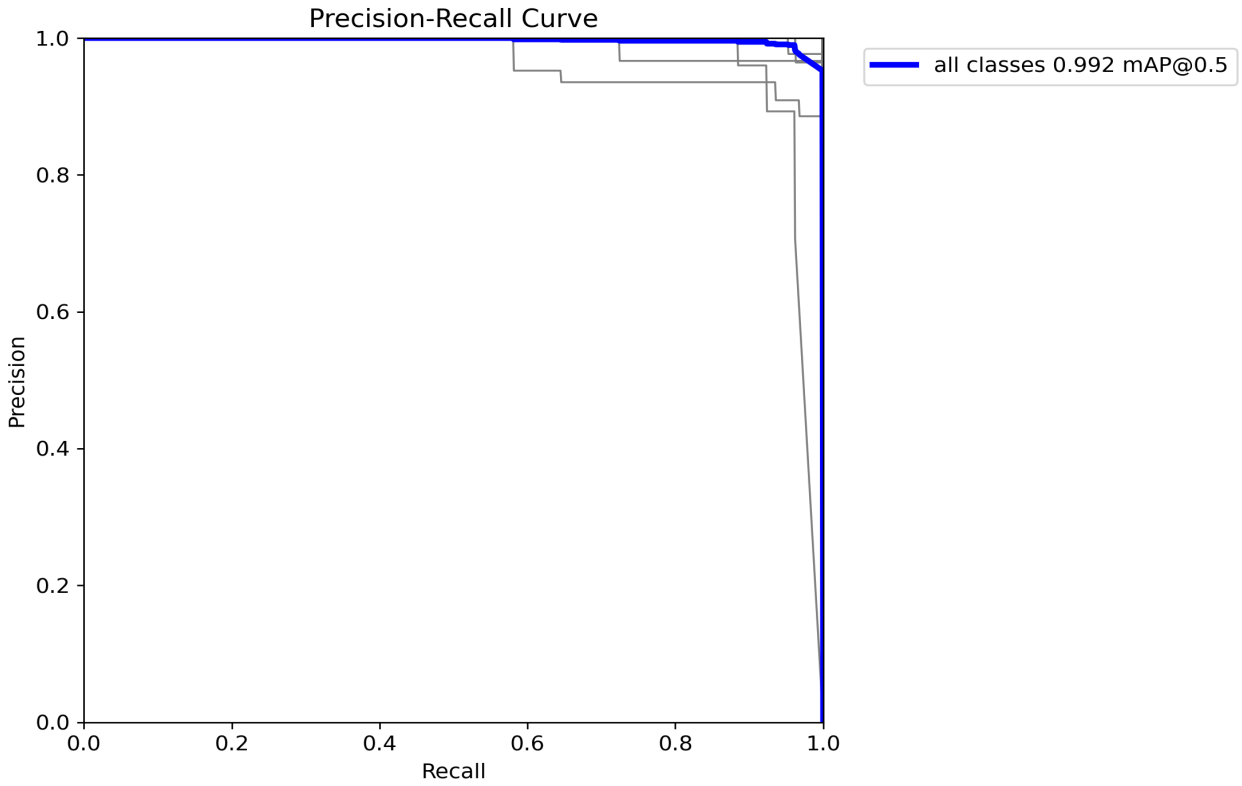
YOLOv8 Results



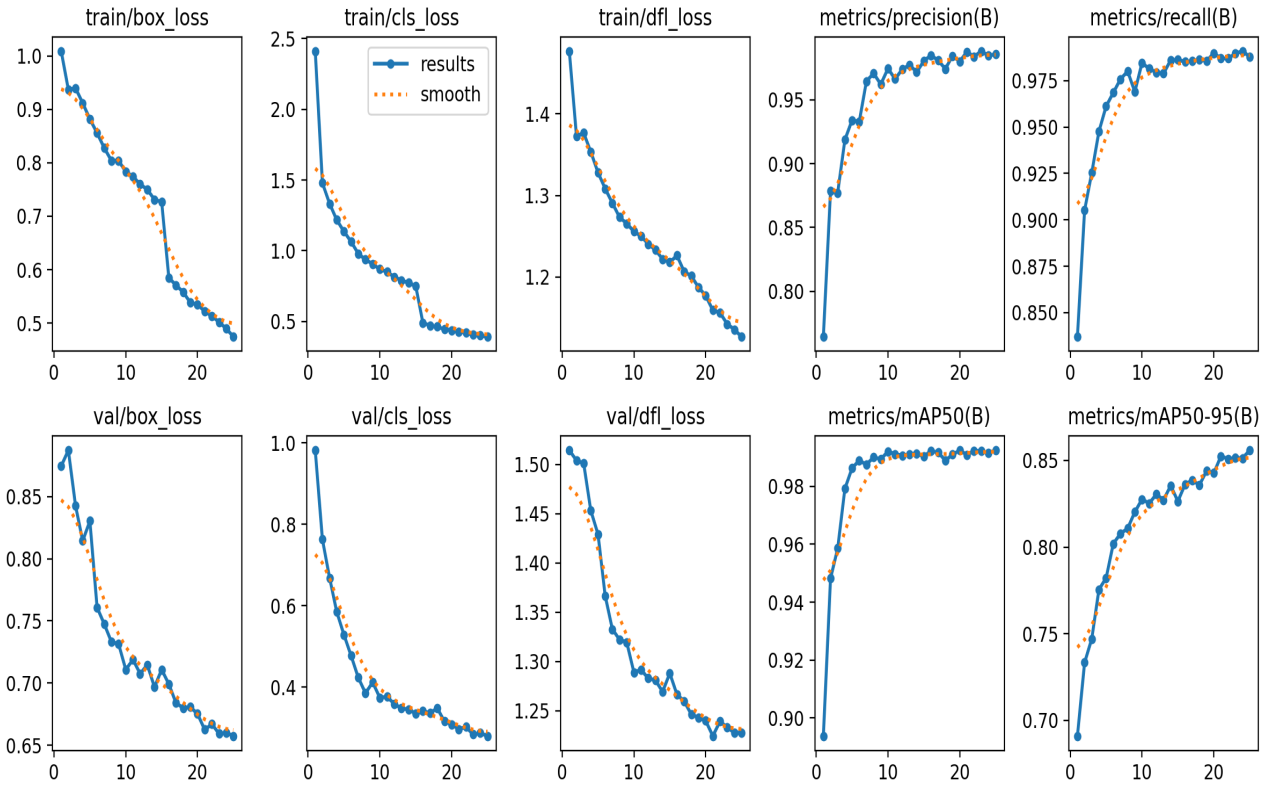
YOLOv8 Results



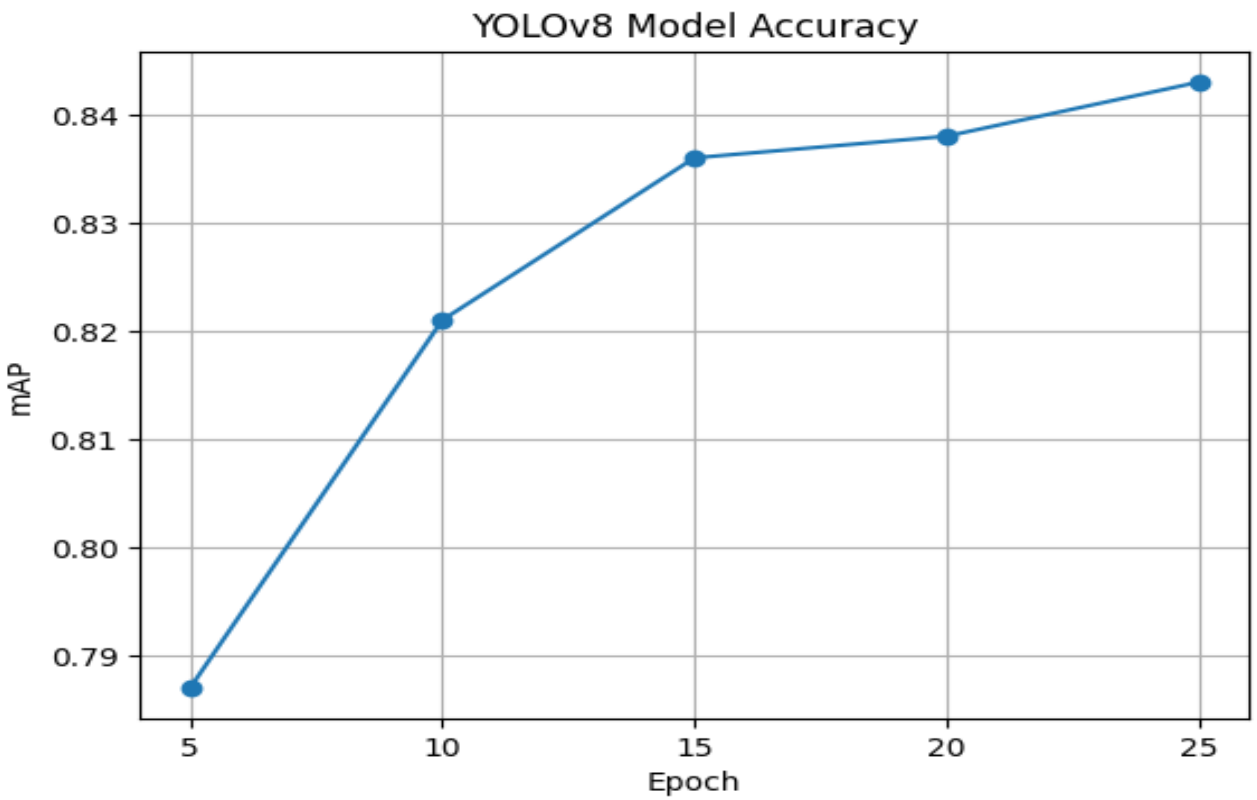
YOLOv8 Results



YOLOv8 Results

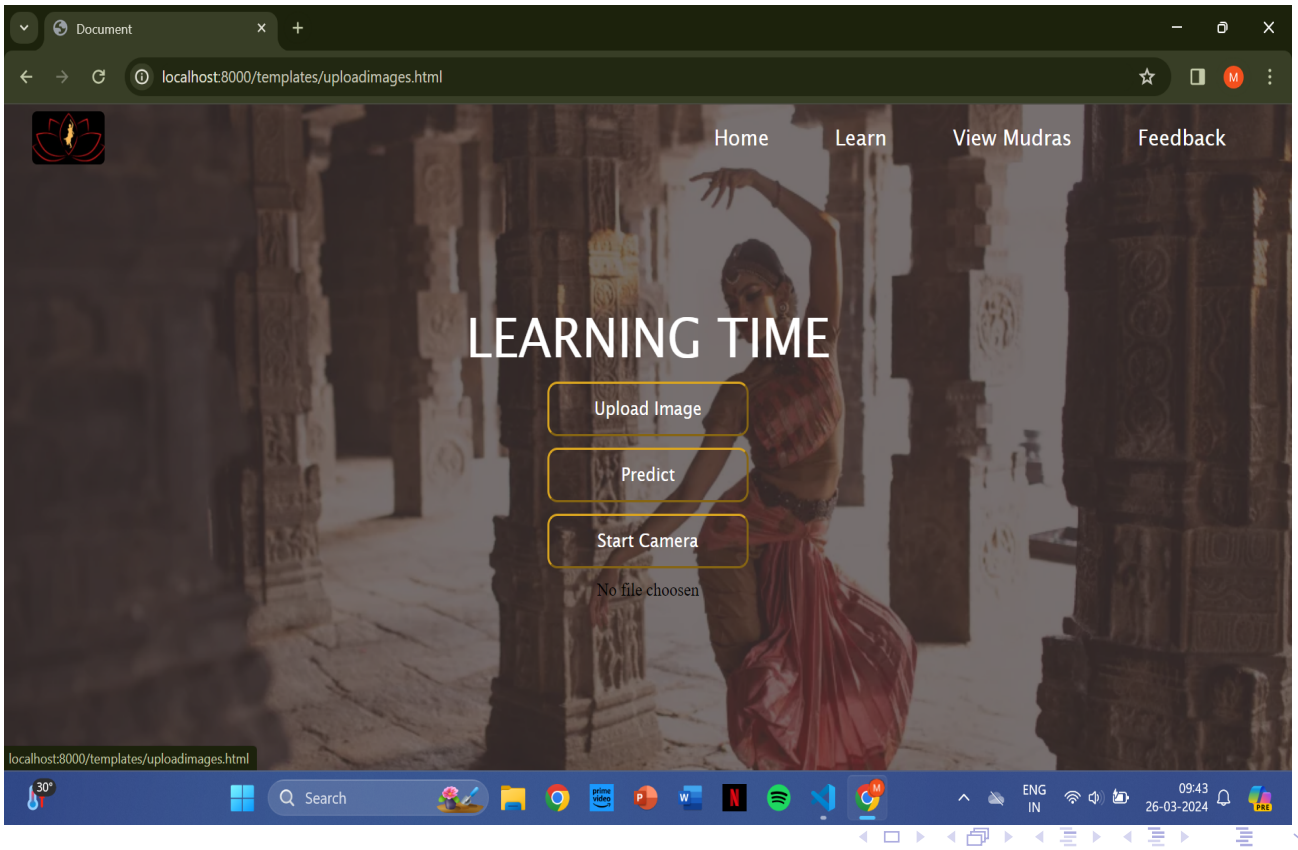


YOLOv8 Results

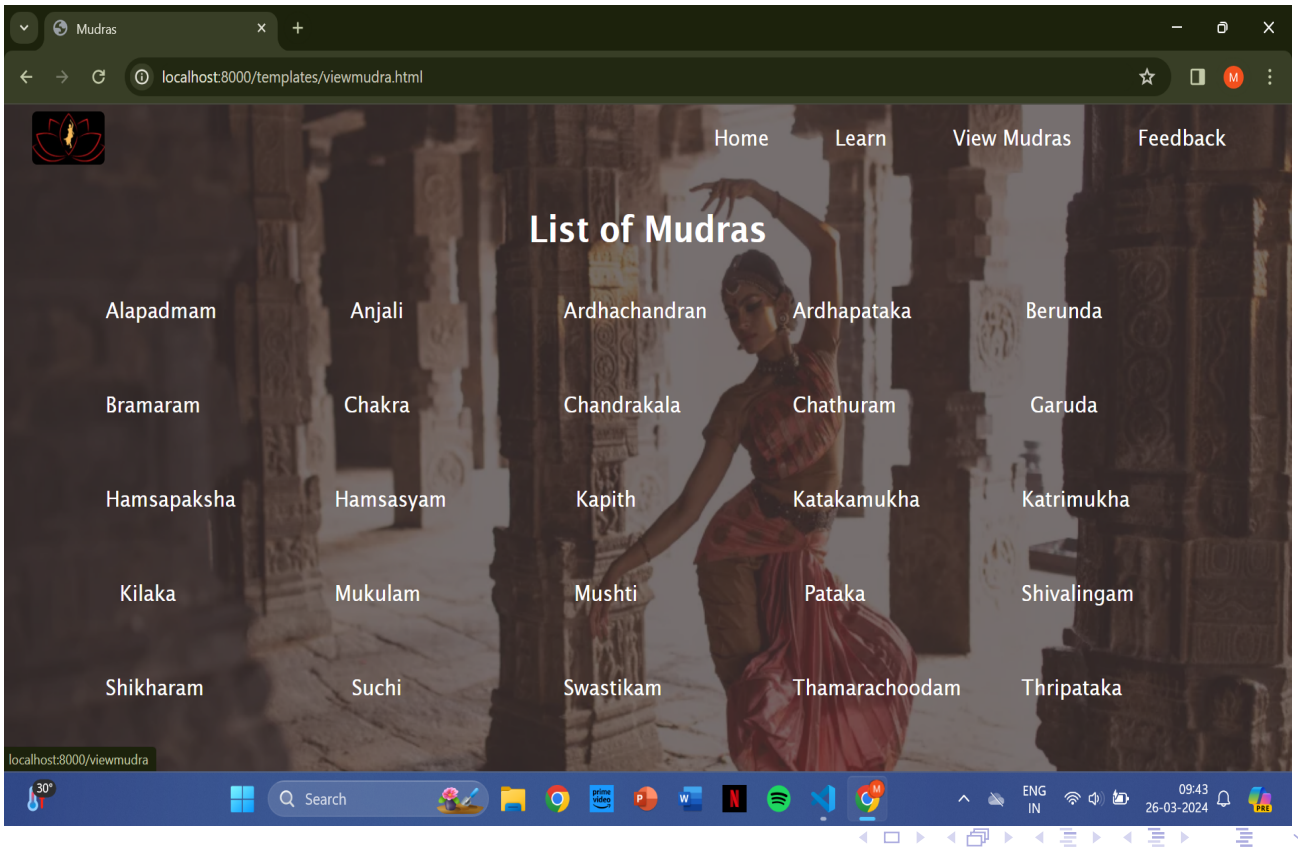


Results

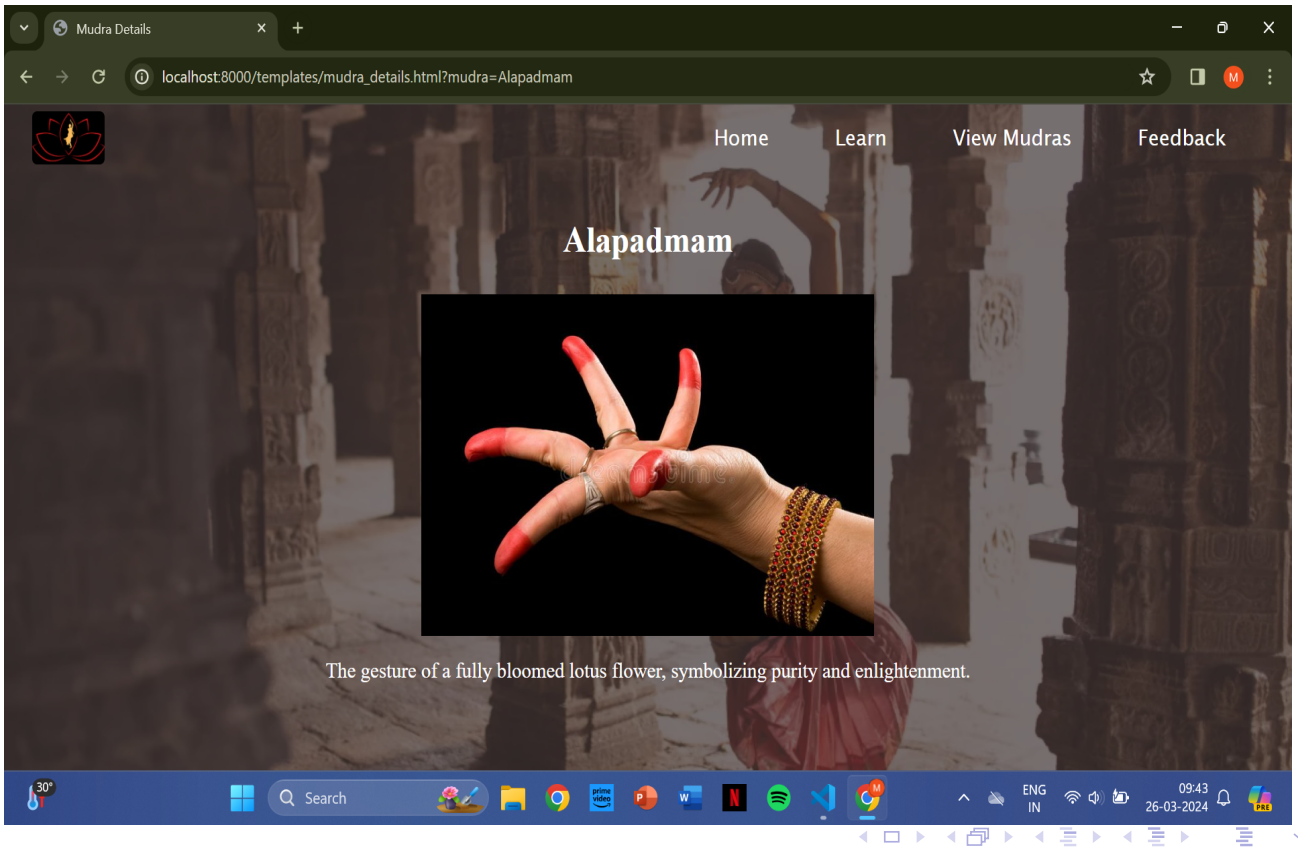
Results



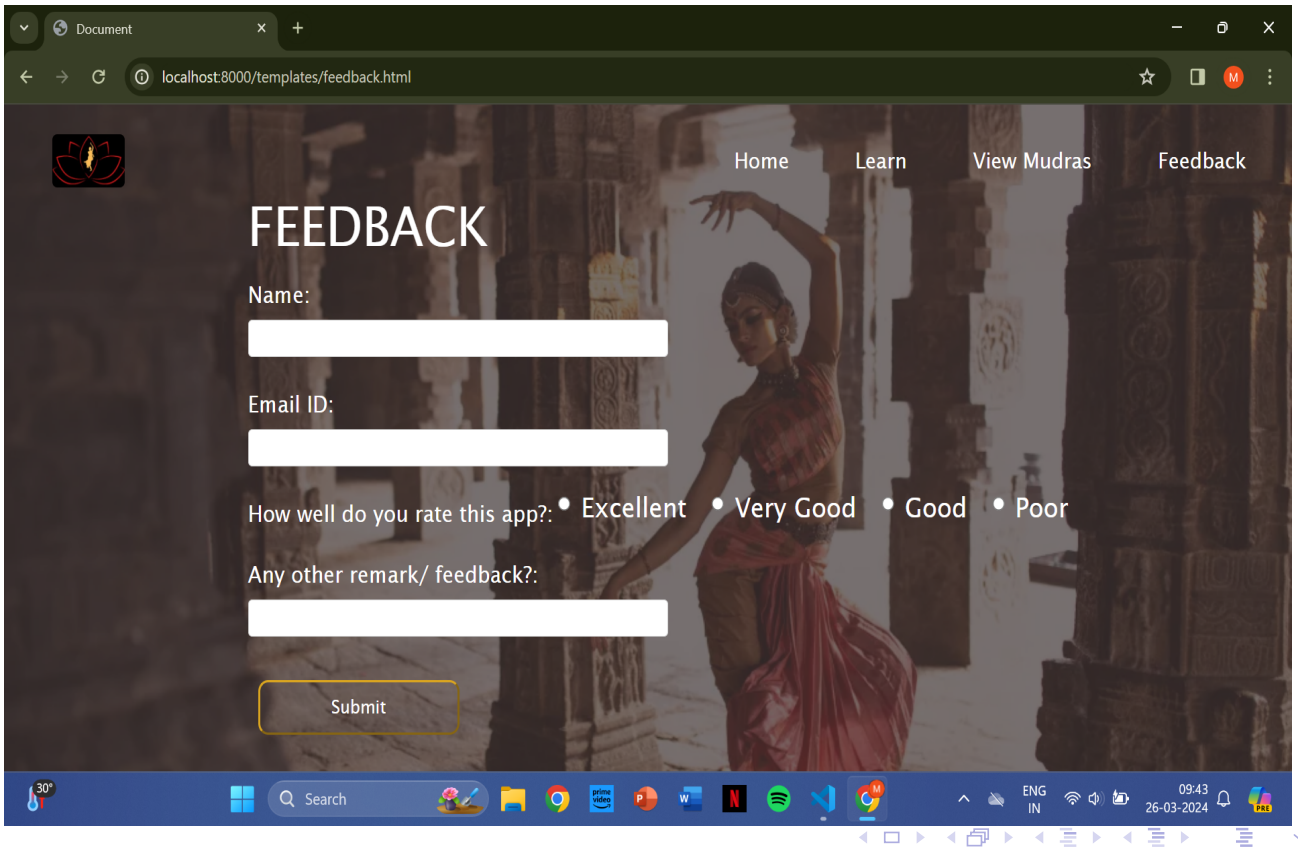
Results



Results



Results



Results

DETAILS OF TRAINING										
MODEL NAME	NO: OF EPOCHS	BATCH SIZE	TRAINING TIME	ACCURACY	LOSS	NO OF CLASSES	AUGMENTED OR NOT	SEGMENTED OR NOT	SYSTEM USED	
CNN	50	32	1 hr	45%	0.887	**5**	Non-Augmented	Segmented	Laptop	
VGG 16	10	16	1.5 hrs	93%	0.063	**5**	Non-Augmented	Non-Segmented	Laptop	
VGG 16	10	16	1.5 hrs	91%	0.083	**5**	Non-Augmented	Segmented	Laptop	
Staged VGG16	10	8	22 mins	99% sub mudra 1 98% for submudra2 others all in 22% range	0.9816	**50**	Non-Augmented	Non-Segmented	TinkerSpace	
Staged VGG16	20	32	23 mins	99% sub mudra 1 98% for submudra2 others all in 22% range	0.0289	**50**	Non-Augmented	Non-Segmented	TinkerSpace	
Staged VGG16	20	32	51 mins	99% sub mudra 1 100% for submudra2 others all in 22% range	0.0025	**50**	Augmented	Non-Segmented	TinkerSpace	
Staged VGG16	50	32	1.5 hr	22%	1.66	**50**	Augmented	Non-Segmented	TinkerSpace	
Staged VGG16	500	32	15 hrs	19%	1.8	**50**	Non-Augmented	Non-Segmented	TinkerSpace	
Staged VGG16	10	8	30 mins	34%	1.8404	**50(in 5 classes)**	Non-Augmented	Non-Segmented	TinkerSpace	
Staged VGG16	10	32	25 mins	99.8% for 25 classes	0.0318	**25**	Non-Augmented	Non-Segmented	TinkerSpace	
YOLOV8	10	25	6 mins	82.60%	0.9	**25**	Non-Augmented	Non-Segmented	Colab	
YOLOV8	30	16	15 mins	83.40%	0.41	**25**	Non-Augmented	Non-Segmented	Colab	
YOLOV8	20	16	1.2 hrs	85%	0.47	**25**	Augmented	Non-Segmented	Colab	
YOLOV8	150	16	52 mins	84.60%	0.22	**25**	Non-Augmented	Non-Segmented	Colab	
YOLOV8	25	16	32 mins	84.30%		**25**	Non-Augmented	Non-Segmented	Colab	
YOLOV5	25	16	45 mins	79%		**25**	Augmented	Non-Segmented	Colab	
YOLOV5	50	16	90 mins	81.50%		**25**	Augmented	Non-Segmented	Colab	
YOLOV5	100	16	2.7 hours	82.30%		**25**	Augmented	Non-Segmented	Colab	

Future Scope

- Enhancing interactive real time features.
- Implementing extended training with higher epoch counts to improve accuracy.
- Incorporating correction mechanisms to improve the overall system efficiency for YOLOv8.
- Extend the version for more number of mudras.

Task Distribution

- NEKHA
  Trained segmented and non-augmented color images using VGG16.
  Trained YOLOv8 model.
  Created user interface.
- POOJA
  Trained non-segmented and augmented color images using VGG16.
  Trained YOLOv5 model.
  Connection of backend and frontend.

Task Distribution

- SANDRA
  Trained non-segmented and non-augmented color images using VGG16.
  Testing and training of YOLOv5 model.
  Created user interface.
- SAYUJYA
  Trained segmented and augmented color images using VGG16.
  Testing and training of YOLOv8 model.
  Connection of backend and frontend.

Conclusion

Utilizing both YOLOv8 and YOLOv5 for training a model to recognize 25 distinct dance mudras, achieving respective accuracies of 86% and 85%, represents an innovative approach to integrating cutting-edge object detection algorithms into the realm of dance education.

This approach not only showcases the adaptability and effectiveness of advanced computer vision techniques but also underscores their potential to enhance traditional teaching methods within artistic disciplines.

References

- Herbaz, N., El Idrissi, H. and Badri, A., 2022. A Moroccan Sign Language Recognition Algorithm Using a Convolution Neural Network. Journal of ICT Standardization, 10(3), pp.411-426.
- Jyotishman Bora, Saine Dehingia, Abhijit Boruah, Anuraag Anuj Chetia, DikhitGogoi,Real-time Assamese Sign Language Recognition using MediaPipe and Deep Learning,Procedia Computer Science,Volume 218,2023,Pages 1384-1393,ISSN 1877-0509, https://doi.org/10.1016/j.procs.2023.01.117.
- Ahmad Zaki Shukor, Muhammad Fahmi Miskon, Muhammad Herman Jamaluddin, Fariz bin Ali@Ibrahim, Mohd Fareed Asyraf, Mohd Bazli bin Bahar,A New Data Glove Approach for Malaysian Sign Language Detection,Procedia,Computer Science,Volume 76,2015,Pages 60-67,ISSN 1877-0509,https://doi.org/10.1016/j.procs.2015.12.276.

References

- M. Mohandes, M. Deriche and J. Liu, "Image-Based and Sensor-Based Approaches to Arabic Sign Language Recognition," in IEEE Transactions on Human-Machine Systems, vol. 44, no. 4, pp. 551-557, Aug. 2014, doi: 10.1109/THMS.2014.2318280.
- J. Rekha, J. Bhattacharya and S. Majumder, "Shape, texture and local movement hand gesture features for Indian Sign Language recognition," 3rd International Conference on Trendz in Information Sciences Computing (TISC2011), Chennai, India, 2011, pp. 30-35, doi: 10.1109/TISC.2011.6169079.
- M. K. Bhuyan, M. K. Kar and D. R. Neog, "Hand pose identification from monocular image for sign language recognition," 2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA), Kuala Lumpur, Malaysia, 2011, pp. 378-383, doi: 10.1109/ICSIPA.2011.6144163.

References

- Tang, Xianlun Yan, Zhenfu Peng, Jiangping Hao, Bohui Wang, Huiming Li, Jie. ,2021. Selective spatiotemporal features learning for dynamic gesture recognition. Expert Systems with Applications. 169. 114499. 10.1016/j.eswa.2020.114499.
- Ji, S., Xu, W., Yang, M., Yu, K. (2012). 3D convolutional neural networks for human action recognition. IEEE Transactions on Pattern Analysis and Machine Intelligence, 35, 221–231.
- Simonyan, K., Zisserman, A. (2014a). Two-stream convolutional networks for action recognition in videos. In Advances in neural information processing systems (pp. 568-576).
- Miao, Q., Li, Y., Ouyang, W., Ma, Z., Xu, X., Shi, W., Cao, X. (2017). Multimodal gesture recognition based on the resc3d network. In Proceedings of the IEEE International Conference on Computer Vision Workshops (pp. 3047–3055).

Status of Paper Publication

- Communicated the paper to the International Journal of Computer Vision and Robotics.

Appendix B: Vision, Mission, Programme Outcomes and Course Outcomes

Vision, Mission, Programme Outcomes and Course Outcomes

Institute Vision

To evolve into a premier technological institution, moulding eminent professionals with creative minds, innovative ideas and sound practical skill, and to shape a future where technology works for the enrichment of mankind.

Institute Mission

To impart state-of-the-art knowledge to individuals in various technological disciplines and to inculcate in them a high degree of social consciousness and human values, thereby enabling them to face the challenges of life with courage and conviction.

Department Vision

To become a centre of excellence in Computer Science and Engineering, moulding professionals catering to the research and professional needs of national and international organizations.

Department Mission

To inspire and nurture students, with up-to-date knowledge in Computer Science and Engineering, ethics, team spirit, leadership abilities, innovation and creativity to come out with solutions meeting societal needs.

Programme Outcomes (PO)

Engineering Graduates will be able to:

1. Engineering Knowledge: Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.

2. Problem analysis: Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.

3. Design/development of solutions: Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.

4. Conduct investigations of complex problems: Use research-based knowledge including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.

5. Modern Tool Usage: Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modeling to complex engineering activities with an understanding of the limitations.

6. The engineer and society: Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal, and cultural issues and the consequent responsibilities relevant to the professional engineering practice.

7. Environment and sustainability: Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. Ethics: Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.

9. Individual and Team work: Function effectively as an individual, and as a member or leader in teams, and in multidisciplinary settings.

10. Communication: Communicate effectively with the engineering community and with society at large. Be able to comprehend and write effective reports documentation. Make effective presentations, and give and receive clear instructions.

11. Project management and finance: Demonstrate knowledge and understanding of engineering and management principles and apply these to one's own work, as a member and leader in a team. Manage projects in multidisciplinary environments.

12. Life-long learning: Recognize the need for, and have the preparation and ability to engage in independent and lifelong learning in the broadest context of technological change.

Programme Specific Outcomes (PSO)

A graduate of the Computer Science and Engineering Program will demonstrate:

PSO1: Computer Science Specific Skills
The ability to identify, analyze and design solutions for complex engineering problems in multidisciplinary areas by understanding the core principles and concepts of computer science and thereby engage in national grand challenges.

PSO2: Programming and Software Development Skills
The ability to acquire programming efficiency by designing algorithms and applying standard practices in software project development to deliver quality software products meeting the demands of the industry.

PSO3: Professional Skills
The ability to apply the fundamentals of computer science in competitive research and to develop innovative products to meet the societal needs thereby evolving as an eminent researcher and entrepreneur.

Course Outcomes (CO)

Course Outcome 1: Model and solve real world problems by applying knowledge across domains (Cognitive knowledge level: Apply).

Course Outcome 2: Develop products, processes or technologies for sustainable and socially relevant applications (Cognitive knowledge level: Apply).

Course Outcome 3: Function effectively as an individual and as a leader in diverse teams and to comprehend and execute designated tasks (Cognitive knowledge level: Apply).

Course Outcome 4: Plan and execute tasks utilizing available resources within timelines, following ethical and professional norms (Cognitive knowledge level: Apply).

Course Outcome 5: Identify technology/research gaps and propose innovative/creative solutions (Cognitive knowledge level: Analyze).

Course Outcome 6: Organize and communicate technical and scientific findings effectively in written and oral forms (Cognitive knowledge level: Apply).