HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION

FACULTY FOR HIGH QUALITY TRAINING

COMPUTER ENGINEERING TECHNOLOGY

SENIOR PROJECT 2

ELECTRONIC DEVICE CONTROL USING HAND GESTURE RECOGNITION SYSTEM

Student: **DO DUY TRUONG**
MSSV: 20119174

Advisor: Ph.D.: HUYNH THE THIEN

HO CHI MINH CITY, 07/2023

HO CHI MINH UNIVERSITY OF TECHNOLOGY AND EDUCATION
FACULTY FOR HIGH QUALITY TRAINING

SENIOR PROJECT 2

ELECTRONIC DEVICE CONTROL USING HAND GESTURE RECOGNITION SYSTEM

COMPUTER ENGINEERING TECHNOLOGY

Student: **DO DUY TRUONG**
MSSV: 20119174

Advisor: Ph.D.: HUYNH THE THIEN

HO CHI MINH CITY, 07/2023

ACKNOWLEDGMENTS

Mr. Huynh The Thien gave me a lot of advise and support when I was working on my senior project 2 on "Electronic Device Control Using Hand Gesture Recognition System." I'd want to offer my heartfelt appreciation for his invaluable guidance and support along this trip. His knowledge and insights have been crucial in determining the project's direction and outcomes. I am extremely grateful for his persistent support and commitment to assisting me in reaching my academic objectives.

Project Executor
(Sign and clearly write your full name.)

DO DUY TRUONG

DECLARATION

The senior project 2 executor hereby confirms that the project was carried out based on a number of previously referred materials and that no content or outcomes from other projects were copied. All mentioned information has been correctly cited.

Project Executor
(Sign and clearly write your full name.)

DO DUY TRUONG

ABSTRACT

An innovative computer vision research project called "Electronic Device Control Using Hand Gesture Recognition System" aims to use hand gestures to control electrical gadgets. This cutting-edge strategy challenges conventional control strategies and seeks to develop a smooth and natural connection between people and technology. The suggested system makes use of hand gestures to streamline control and improve user experience, especially in situations when conventional input techniques are difficult or unworkable.

Through comprehensive testing and assessment, the research examines the creation and efficacy of this gesture recognition system. The architecture of the system uses feature extraction methods and machine learning algorithms that are specifically designed to properly understand hand movements. Examining aspects including identification precision, reaction speed, and robustness to varied environmental circumstances, the system's performance is tested using a variety of hand gestures.

The study's findings demonstrate the hand gesture recognition system's potential as a workable substitute for controlling technological devices. The system exhibits encouraging precision and reactivity, demonstrating its capacity to precisely understand and carry out human wishes. The advancement of user interfaces and human-computer interactions in a variety of situations, such as smart homes, virtual reality settings, and accessibility tools for people with mobility issues, will be greatly impacted by this development. The study's main finding emphasizes the viability of improving technology control using natural hand gestures, creating new opportunities for the seamless incorporation of electronic gadgets into daily life.

Contents

List of Figures

List of abbreviations

Below is the list of abbreviations used in the project.

Abbreviations	**Definition**
ML	Machine Learning
RSA	Radar Signal Analysis
ANN	Artificial Neural Network
CNN	Convolutional Neural Network
SGD	Stochastic Gradient Descent
RGB	Red Green Blue
AI	Artificial Intelligence
GPU	Graphics Processing Unit
JPEG	Joint Photographic Experts Group
GUI	Graphical User Interface
FPS	Frames Per Second
VGG16	Visual Geometry Group 16
MSE	Mean Squared Error
ReLU	Rectified Linear Activation Function
Adam	Adaptive Moment Estimation
MOG2	Mixture of Gaussians background subtraction algorithm

Chapter 1

OVERVIEW

1.1 Introduction

More logical and effective controls are required as the world becomes more dependent on technological technologies. The "Electronic Device Control Using Hand Gesture Recognition System" project presents a revolutionary method. This project seeks to transform how humans engage with technology by utilizing the possibilities of hand gestures. A complex hand gesture recognition system will be created using cutting-edge computer vision and machine learning techniques, allowing gadgets to comprehend and react to a variety of movements. The precision and responsiveness of the system will be carefully examined in this study, perhaps opening the door for more realistic and immersive electronic device control experiences.

1.2 Objective

The "Electronic Device Control Using Hand Gesture Recognition System" project's main goal is to create and put into use a novel system that allows users to control electronic devices with hand gestures. The goal of this project is to properly convert a range of hand gestures into commands for electrical devices by utilizing computer vision and machine learning techniques. The main goal is to develop an interaction paradigm that is intuitive, responsive, and user-friendly and that improves how people interact with

technology. This project seeks to revolutionize human-computer interaction by developing a powerful hand gesture detection system, with possible applications in a number of areas, including accessibility, smart homes, and virtual reality settings.

1.3 Research situation

Through the creation of a sophisticated hand gesture detection system specifically designed for electronic device control, this research effort seeks to close these gaps. The project aims to develop a system that properly reads a variety of hand gestures by combining computer vision and machine learning approaches, ultimately enhancing user engagement with electronics. The research aims to determine the efficacy and applicability of the proposed system through systematic tests and thorough assessments. With applications ranging from smart homes to virtual reality and beyond, this inquiry has the potential to shed light on the promise and constraints of hand gesture-based device control.

1.4 Research methodology

I employed data collecting, experimenting, and synthesizing the outcomes of empirical analysis to accomplish the topic's objective.

- Approaches to gathering data:

  - Applying the observational approach: I performed a live observation utilizing a data set generated by the camera.

- Testing accessibility:

  - I create and deploy on Windows for system design and execution.

- Analysis techniques, a summary of the experience:

  - Efficacy evaluation: Using the average accuracy of the test findings, I rate the test's effectiveness.

  - Experience synopsis: Based on the findings of the investigation, I apply my previous work on AI projects.

1.5 Content structure

My report for the senior project 2 will consist of 5 chapters.

- Chapter 1: Overview
- Chapter 2: Theoretical basis
- Chapter 3: System design
- Chapter 4: Result
- Chapter 5: Conclusion

Chapter 2

THEORETICAL BASIS

2.1 Deep Learning

2.1.1 Artificial Neural Network

The VGG16 model's design is fundamentally based on the Artificial Neural Network (ANN) idea. The key convolutional neural network (CNN), VGG16, gets its power from the ANN ideas. ANNs, which are the foundation of artificial intelligence and excel in tasks involving recognition, classification, prediction, and data processing, are inspired by the neural networks of the human brain.

The ANN structure in the VGG16 model refers to a network of artificial neurons that are methodically linked. Each computer unit or artificial neuron functions as a node in the network. It takes information from other neurons as inputs, activates the information, and then transfers the results to neurons downstream. These connections store weights that ANNs learn to change adaptively, allowing the network to approach desired results. The training process involves feeding the ANN a labeled training dataset with corresponding input and output pairs. During training, the ANN strives to minimize the discrepancy between predicted outputs and actual outputs by iteratively refining the weights associated with connections. The VGG16 model, with its 16 layers, integrates this ANN architecture to effectively extract intricate features from images, demonstrating its proficiency in tasks such as image classification.

An labeled training dataset with associated input and output pairs is fed to the ANN during training. By iteratively adjusting the weights connected to connections, the ANN aims to reduce the difference between anticipated outputs and actual outputs throughout training. With its 16 layers, the VGG16 model incorporates this ANN design to successfully extract complex information from pictures, showcasing its competence in tasks like image categorization.

Upon successful training, the ANN within VGG16 can generalize its learned representations to novel data. In the diagram below, a simplified illustration of an ANN with multiple inputs, two hidden layers, and three outputs is presented:

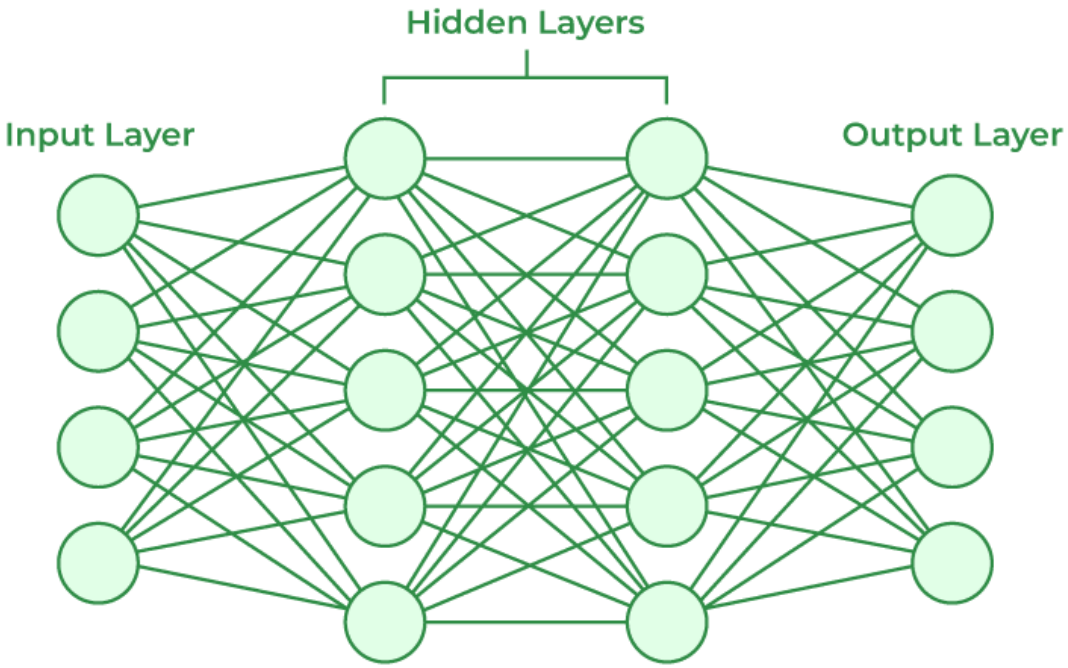

Figure 2.1: Artificial Neural Network

2.1.2 Convolutional Neural Network

An artificial neural network called a convolutional neural network (CNN) is made particularly to process and interpret organized grid-like input, such pictures and sounds. In several computer vision applications, CNN has demonstrated to be quite efficient.

In order to extract local characteristics from the input, CNN has a unique design known as a "convolutional layer," which employs tiny filters. In order to create a feature map, these filters traverse over the picture while computing local responses. This method aids the model's learning of regional characteristics like corners and edges.

In order to minimize the amount of features and improve spatial invariance to positional shifts, CNN frequently uses pooling layers after the convolutional layers. Fully linked layers that aggregate the retrieved characteristics and carry out the final classifications come next.

CNN can automatically learn complicated characteristics from input and adjust to spatial changes because to this particular design. It has thus found extensive use in a variety of computer vision tasks, including picture classification, object identification, face recognition, and many more applications. A convolutional neural network with numerous layers is shown in the diagram below.

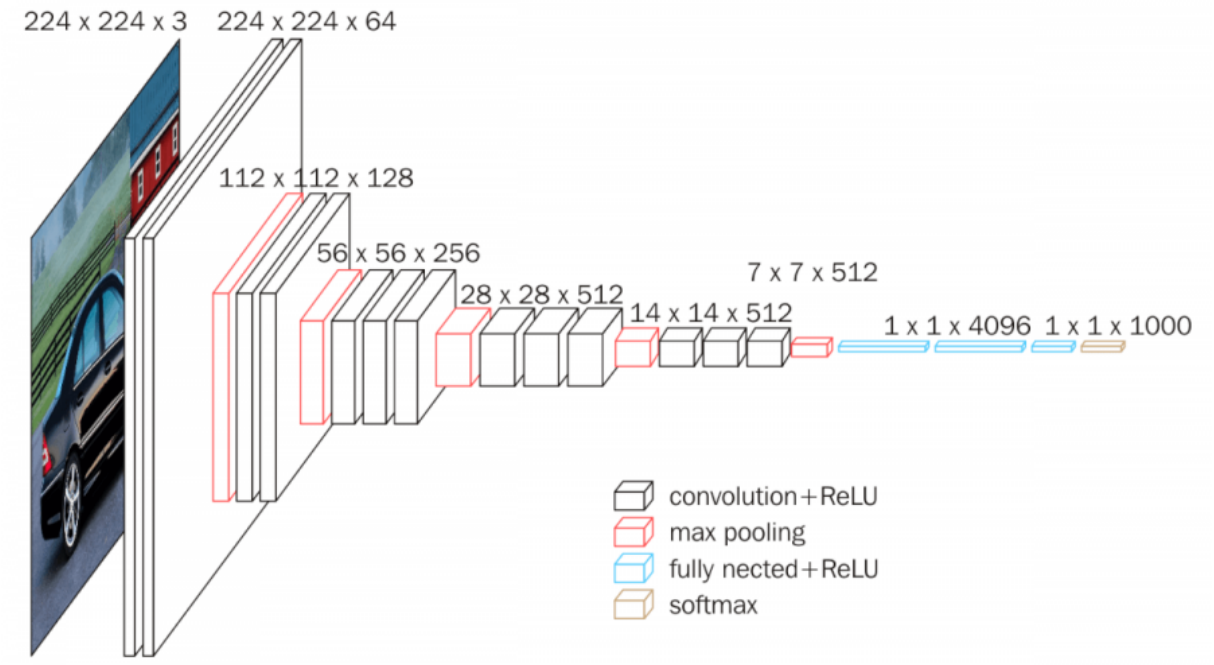

Figure 2.2: Convolutional Neural Network.

2.2 Object detection

2.2.1 What is object detection ?

The process of locating and categorizing things in an image, a video, or other input data is known as object detection. The objective of object detection is to identify and comprehend various items in a scene or context.

2.2.2 Types of object detection

The technique of identifying and categorizing items in 2D space, such as static photographs, is known as 2D object recognition. It often entails pinpointing the location, character, and attributes of objects in a picture.

The technique of finding and categorizing items in three dimensions is known as 3D object recognition. It entails figuring out an object's form, structure, and spatial features. Applications including computer vision, virtual reality, and robot control frequently employ 3D object identification.

The practice of quickly and continually identifying and categorizing items in real-time is known as real-time object recognition. Applications like driverless cars, security monitoring, facial recognition, and object detection in movies all frequently make use of it.

2.3 VGG16 architecture

A key convolutional neural network (CNN) architecture in the field of computer vision is the VGG16 (Visual Geometry Group 16) network. This model's popularity is a result of its simple yet effective design. The VGG16 algorithm is designed primarily for feature extraction from pictures and carries out operations like image categorization.

A total of 16 layers make up the VGG16 architecture, including 13 convolutional layers and 3 fully linked layers. The network's central component, the convolutional layers, retrieve information from pictures by running convolutional filters over several input regions. The network's last fully connected layers process the retrieved information and

generate its output.

The constancy of the structure of the convolutional layers in the VGG16 architecture is a significant feature. All layers use 2x2 max-pooling for subsampling and 3x3 filters with strides of 1 and 3, respectively. With the help of the ultimate completely linked layers and the use of smaller filters, the network is able to learn complex characteristics seen in photos.

The VGG16 network has competed in picture classification challenges and produced outstanding outcomes on test datasets including the ImageNet dataset. VGG16 continues to offer a substantial basis for research and development of advanced convolutional neural network designs in spite of having a relatively high number of parameters in comparison to succeeding models.

2.4 Working Principle of VGG16 Network

The VGG16 (Visual Geometry Group 16) network's hierarchical feature extraction procedure, carried out through a number of convolutional and fully connected layers, is the basis of its operation. This architecture is especially well suited for tasks like image classification since it is made to efficiently capture complicated patterns inside photos.

1. **Input Layer**: The input layer, which is the first step in the process, is where the raw picture data is entered. This input's size corresponds to the size of the photos being processed.
2. **Convolutional Layers**: A stack of convolutional layers defines the VGG16. These layers scan the input picture with convolutional filters to extract different characteristics including edges, textures, and forms. The network may learn ever-more complicated visual representations by repeatedly applying these filters in subsequent layers.
3. **Activation Functions**:In order to inject non-linearity into the network, an activation function—typically a Rectified Linear Unit (ReLU)—is applied element-wise after each convolutional operation. This improves the network's capacity to identify intricate data linkages.

4. **Max Pooling Layers**: Max pooling layers are added in order to downsample the feature maps and lessen computational complexity. These layers preserve the most important data by selecting the maximum value from each local area of the feature map.
5. **Fully Connected Layers**: The convolutional and pooling layers are followed by a series of fully linked layers in the network. These layers serve as classifiers, deriving predictions for distinct classes based on the retrieved characteristics.
6. **Softmax Activation**: A softmax activation function, which transforms the raw output scores into probabilities, is frequently installed in the last fully linked layer. These probabilities show how confident the network is in each class.
7. **Output Layer**: The output layer produces the final classification results based on the input image's predicted class, which is the class with the highest probability.

The deep design of the VGG16 network, which consists of 16 layers in total, enables it to recognize complex hierarchical characteristics in pictures with great accuracy. The network can learn and detect patterns at different levels of abstraction because to the hierarchical encoding of features, which helps the network correctly categorize a variety of pictures.

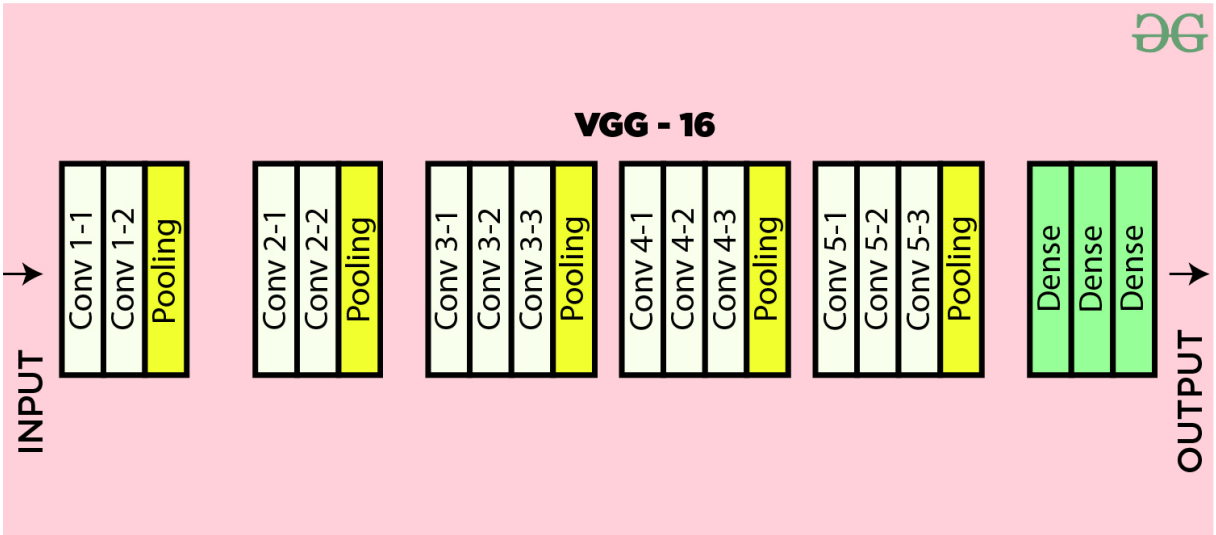

Figure 2.3: VGG16 architectural diagram

2.5 Data Preprocessing

Preparing the raw picture dataset for the hand gesture recognition system's efficient training is a crucial part of data preparation. This procedure involves a number of steps designed to structure and standardize the input data in order to provide the best model training results.

2.5.1 Image Resize

Each image in the collection is downsized to 224x224 pixels in order to preserve consistency and enable quick processing. In addition to guaranteeing consistency among photos, this downsizing step significantly lowers the computational demands made during model training.

2.5.2 Formatting to Be Network-Ready

Images are transformed into a format that the neural network can accept as input. Initial photos, usually in the JPEG or PNG format, are converted into numerical arrays so that the model can process them. Additionally, the grayscale pictures are transformed into three-channel representations since the VGG16 model requires input images with three color channels (RGB).

2.5.3 Normalization

To normalize the pixel values of the photographs, normalization is done. In this stage, the pixel values are scaled to a range between 0 and 1. Normalization makes it possible for the model to successfully learn from the data and helps to reduce numerical instability during model training.

2.5.4 Label Encoding

Each image's related gesture labels must be transformed into numerical values the neural network can understand. Each type of gestures is given a special numerical iden-

tification through the establishment of a label mapping. During model training, this encoding makes the categorization process easier.

2.5.5 Sorting Out Data

The data is categorized into input features (X) and associated target labels (Y) once the pictures have been preprocessed and tagged. The goal labels are the numerical representations of the gesture categories, whereas the input features are the processed picture arrays.

Data preparation is a fundamental step that guarantees the input data is organized and ready for the hand gesture recognition system's training. The data preparation procedure provides a standardized and well-prepared dataset that is ready to be fed into the neural network for training by scaling photos, translating to the appropriate formats, normalizing pixel values, maybe supplementing data, and encoding labels.

2.6 Data Labeling and Mapping

Data labeling and mapping are essential for converting the semantic meaning of gesture categories into a numerical representation that the neural network can comprehend. This stage is crucial for transforming the identified hand gestures associated with each image into a format suitable for classification.

- **Gesture Labeling** Each image in the dataset is associated with a specific hand gesture category, such as 'E' (fi), 'L' (L_), 'F' (ok), 'V' (pe), or 'B' (pa). These labels provide human-readable identification for the gestures present in the images.
- **Label Mapping** To enable the neural network to work with gesture categories, a label mapping is established. Each unique gesture category is assigned a numerical identifier that serves as the network's way of recognizing and categorizing the gestures. For instance, the gesture 'E' is mapped to the numerical value 0, 'L' to 1, 'F' to 2, 'V' to 3, and 'B' to 4.

- **One-Hot Encoding** Numerical labels undergo one-hot encoding to facilitate categorical classification. This technique transforms numerical gesture identifiers into binary vectors, where only one element is '1' (hot) to represent each gesture type, while all other elements are '0' (cold), corresponding to the numerical gesture identifiers.
- **Label-Encoded Data** Following labeling and mapping, the dataset's labels are converted into numerical values using the designated mapping. This label-encoded data serves as the target variable that the neural network aims to predict during the training phase.

In summary, data labeling and mapping involve the translation of human-readable gesture labels into numerical representations, enabling effective communication between the gesture recognition system and the neural network. Label mapping and one-hot encoding ensure accurate recognition and classification of the various hand gestures present in the dataset.

2.7 Transfer Learning and Fine-Tuning

The VGG16 model needs to undergo modifications using transfer learning and fine-tuning techniques to suit the specific objective of hand gesture detection. These strategies expedite training and enhance the model's capability to capture relevant features by leveraging pre-trained models.

- **Transfer Learning:**

  Transfer learning entails extracting shared features from images using a pre-trained model that has been trained on a substantial dataset, often ImageNet. The convolutional layers of the VGG16 model have learned to recognize various shapes, textures, and patterns present in images. Instead of starting from scratch, we can utilize this pre-trained model as a starting point to take advantage of the learned features.
- **Feature Extraction:** The initial layers of the pre-trained VGG16 model are employed as feature extractors in transfer learning. These layers capture low-level features such as edges, textures, and basic shapes. By adjusting the parameters of these

layers and retrieving their output, we establish a foundation of valuable features for our specific task.

- **Fine-Tuning:** To adapt the model for the new task of hand gesture recognition, the weights of the model's subsequent layers need to be adjusted. Fine-tuning focuses on training the deeper layers to identify high-level elements specific to hand motions, as the early layers have already captured broad aspects.
- **Adding New Layers:** During fine-tuning, new fully connected layers are added on top of the pre-trained VGG16 model. These layers function as the classification head, learning to distinguish between various types of hand gestures using the retrieved features.
- **Freezing Pre-Trained Layers:** During fine-tuning, the weights of the pre-trained layers are often frozen. This prevents forgetting the general features they have learned, as these layers remain unchanged throughout training. Only the newly added layers are trained to recognize subtle hand motions.
- **Task-Specific Training:** The refined model is then trained using the labeled hand gesture dataset. The goal of task-specific training is to accurately categorize hand gestures into the mapped categories by adjusting the parameters of the newly added layers.

In conclusion, transfer learning and fine-tuning harness the capabilities of the pre-trained VGG16 model for the task of hand gesture detection. By utilizing pre-learned features and fine-tuning them for the specific task, we develop an efficient and effective model for hand gesture recognition and device control.

2.8 Model Compilation and Training

Configuring the model's learning process, choosing appropriate optimization techniques, and repeatedly updating the model's parameters to minimize the classification error are all steps in the creation and training of the hand gesture recognition model. The processes used to create and train the model successfully are described in this section.

- **Compilation:** The optimization algorithm and loss function are defined during model compilation. Categorical cross-entropy is a popular loss function for multi-class classification applications like hand gesture recognition. It gauges the discrepancy between real one-hot encoded labels and anticipated probability. The Adam optimizer, a stochastic gradient descent (SGD) version, is frequently used because of its flexible learning rates and quick convergence.
- **Early Stopping:** Early stopping is used to stop overfitting. If the model's performance on a validation dataset pauses or even worsens during training, training is interrupted. This enhances the model's capacity to generalize by preventing the model from learning noise from the training data.
- **Model Checkpoint:** During training, a model checkpoint is created to save the model's weights. As a result, it is possible to resume using the top-performing model without having to retrain in the event of unforeseen disruptions or for later evaluations.
- **Batch Size and Epochs:** The number of training samples utilized in each iteration, known as the batch size, is chosen to strike a compromise between convergence speed and processing economy. How many times the full training dataset is processed during training is determined by the number of epochs. The learning phase of the model is optimized using these hyperparameters.
- **Training:** The combined setup, labeled hand gesture training data, and validation data are used to train the model. The model's weights and biases are modified during training in order to reduce the stated loss function. Multiple iterations (epochs) of the training process are performed until convergence or early termination criteria are satisfied.

2.9 Callbacks

Callbacks are essential to the training process because they allow you to take charge and interact with training based on specific occurrences or successful results. They are crucial for tracking and enhancing the model's performance as it is being trained.

- **Early Stopping**: After each epoch, this callback evaluates the model's performance against the validation set. Training can be terminated early to prevent overfitting if there is no discernible progress over a string of epochs. By doing this, it is made sure that the model is only trained until it performs at its peak on the validation set.
- **Model Checkpoint**: During training, this callback aids in storing the model's weights after each epoch. This makes it simple to keep the model iteration that performs the best. The saved weights can be utilized to build a well-trained model version that can be used if there are any problems or if a later evaluation is required.
- **Scheduler for Learning Rate**: This callback enables changing the learning rate while receiving instruction. A timetable or a set of rules may be used to modify the learning pace. This can improve the learning process and the model's capacity for convergence.
- **Custom Callbacks**: You may make your own callbacks to track other training parameters in addition to the standard callbacks. To make sure the model is properly learning, you may, for example, build a callback to monitor the variation of the loss function on the training and validation sets.

The management and enhancement of the model's performance during training depend heavily on callbacks. They aid in maintaining control over the training procedure, guaranteeing the model learns effectively and delivers the best results on the validation set.

2.10 Model Evaluation and Saving

After training the hand gesture recognition model, it is essential to evaluate its performance and save the trained model for future use. This section outlines the steps involved in evaluating the model's effectiveness and ensuring its preservation.

- **Performance Evaluation**: The trained model's performance is evaluated using various metrics such as accuracy, precision, recall, and F1-score. These metrics provide insights into the model's ability to correctly classify hand gestures. The evaluation is performed on a separate test dataset that the model has not seen during training,

ensuring an unbiased assessment of its performance.

- **Confusion Matrix**: The confusion matrix is a valuable tool for understanding the model's classification results. It illustrates the true positive, true negative, false positive, and false negative predictions, offering a clear picture of where the model excels and where it may struggle.
- **Saving the Model**: Once the model's performance is satisfactory, it is crucial to save the trained model for future use. The model's architecture, weights, and configuration are stored in a file format that can be easily loaded and utilized in other applications without the need to retrain from scratch.
- **Deployment**: The saved model can be deployed in various applications requiring real-time hand gesture recognition, such as electronic device control. Its ability to quickly and accurately identify gestures allows it to enhance user experiences and automate interactions.
- **Continuous Monitoring and Improvement**: Even after deployment, continuous monitoring and evaluation of the model's performance are essential. This ensures that any degradation in performance can be addressed promptly, and the model can be fine-tuned as needed to maintain its accuracy.

In conclusion, the evaluation and preservation of the trained hand gesture recognition model are crucial steps in ensuring its reliability and effectiveness. By thoroughly evaluating its performance and saving it for deployment, the model can contribute to enhanced user experiences and automation in electronic device control.

Chapter 3

SYSTEM DESIGN

The full design of the hand gesture recognition and device control system is covered in this chapter. We will look at the system's architecture, data flow, and implementation details for both the training process and the real-time gesture recognition method.

3.1 Architecture Overview

The system architecture is made up of interconnected components that collaborate to perform hand gesture recognition and device control functions. The following are the major components:

- **Data Acquisition**: This component records real-time video frames from the camera and uses them as input for gesture recognition.
- **Training Pipeline**: The training pipeline consists of loading and preparing training data, building and training the deep learning model, and storing the trained model for later use.
- **Gesture Recognition Engine**: The collected video frames are preprocessed during runtime to improve the quality of hand gesture photos. Based on the processed picture, a pre-trained deep learning network predicts the identified gesture.
- **Device Control**: Based on the recognized gesture, the system triggers device control actions using the `pyautogui` library. This enables users to interact with their devices using hand gestures.

- **User Interface**: In real-time, the user interface displays collected video frames, recognized gestures, and associated device control actions to offer visual feedback.

3.2 Training Process

The following steps are included in the training process:

1. **Data Preprocessing**: To build the training dataset, raw pictures are processed, scaled to a consistent size, and transformed to relevant data types.
2. **Model Architecture**: The VGG16 architecture is used to construct a deep learning model. The underlying model's convolutional layers are frozen, and extra fully connected layers are added for gesture recognition.
3. **Transfer Learning**: Using the training dataset, the pre-trained VGG16 base model is loaded with ImageNet weights and fine-tuned for gesture detection.
4. **Model Evaluation**: The validation dataset is used to assess the model's performance. Callbacks for early halting and model checkpointing assist to prevent overfitting and save the best-performing model.
5. **Model Saving**: The trained model is stored to a file and will be used in real-time gesture recognition in the future.

3.3 Real-Time Gesture Recognition

The real-time gesture recognition mechanism consists of the following steps:

1. **Background Capturing**: During startup, a background picture is collected to aid with background subtraction during preprocessing.
2. **Frame Preprocessing**:To isolate the hand region, captured video frames are preprocessed by eliminating the backdrop and applying thresholding.
3. **Gesture Prediction**: The preprocessed frame is input into the trained model, which uses the learnt attributes to predict the detected gesture.
4. **Device Control Action**: The system performs the relevant device control action, such as adjusting the volume or switching apps, based on the identified gesture.

5. **Visual Feedback**: The system gives the user visual feedback in real-time by presenting the processed frames, detected gestures, and triggered device control actions.

3.4 User Interaction

Hand gestures in front of the webcam are used by users to engage with the system. These motions are captured and interpreted by the system, which then translates them into device control actions. This organic and intuitive connection improves the user experience.

3.5 Summary

This chapter went through the system design for the hand gesture recognition and device control application in detail. The architecture, training procedure, real-time gesture detection technology, and user interface have all been thoroughly explored. The results of the experiments will be presented in the next chapter, and the overall performance of the constructed system will be evaluated.

3.6 Code Train Model

The first stage is to build a machine learning model based on pre-labeled gesture data. This approach learns to recognize and categorize different hand gestures, allowing humans to interact with the computer through hand movements.

To begin, we prepare the dataset and do the essential preprocessing steps to assure data quality. Following that, we build a neural network architecture based on the VGG16 model, with minor tweaks to make it suitable for gesture detection. Following that, the model's performance is evaluated using a dataset separated into training and validation subsets.

Below, I present the source code for the model training process:

```
import os
import warnings
import cv2
import keras
import matplotlib.pyplot as plt
import matplotlib.style as style
import numpy as np
import pandas as pd
from PIL import Image
from tensorflow.keras import models, layers, optimizers
from tensorflow.keras.applications import VGG16
from keras.callbacks import EarlyStopping, ModelCheckpoint
from keras.layers import Dense, Dropout, Flatten
from keras.models import Model
from keras.preprocessing import image as image_utils
from keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.utils import to_categorical
from sklearn.metrics import classification_report, confusion_matrix
from sklearn.model_selection import train_test_split
from PIL import ImageFile

ImageFile.LOAD_TRUNCATED_IMAGES = True


# Dinh nghia cac bien

gestures = {"L_": "L", "fi": "E", "ok": "F", "pe": "V", "pa": "B"}

gestures_map = {"E": 0, "L": 1, "F": 2, "V": 3, "B": 4}
```

```
gesture_names = {0: "E", 1: "L", 2: "F", 3: "V", 4: "B"}


image_path = "data"
models_path = "models/saved_model.hdf5"
rgb = False
imageSize = 224


# Ham xu ly anh resize ve 224x224 va chuyen ve numpy array
def process_image(path):
    img = Image.open(path)
    img = img.resize((imageSize, imageSize))
    img = np.array(img)
    return img


# Xu ly du lieu dau vao
def process_data(X_data, y_data):
    X_data = np.array(X_data, dtype="float32")
    if rgb:
        pass
    else:
        X_data = np.stack((X_data,) * 3, axis=-1)
    X_data /= 255
    y_data = np.array(y_data)
    y_data = to_categorical(y_data)
    return X_data, y_data
```

```
# Ham duuyet thu muc anh dung de train
def walk_file_tree(image_path):
    X_data = []
    y_data = []
    for directory, subdirectories, files in os.walk(image_path):
        for file in files:
            if not file.startswith("."):
                path = os.path.join(directory, file)
                gesture_name = gestures[file[0:2]]
                print(gesture_name)
                print(gestures_map[gesture_name])
                y_data.append(gestures_map[gesture_name])
                X_data.append(process_image(path))

            else:
                continue

    X_data, y_data = process_data(X_data, y_data)
    return X_data, y_data


# Load du lieu vao X va Y
X_data, y_data = walk_file_tree(image_path)

# Phan chia du lieu train va test theo ty le 80/20
X_train, X_test, y_train, y_test = train_test_split(
    X_data, y_data, test_size=0.2, random_state=12, stratify=y_data
)
```

```
# Dat cac checkpoint de luu lai model tot nhat
model_checkpoint = ModelCheckpoint(filepath=models_path, save_best_only=True)
early_stopping = EarlyStopping(
    monitor="val_acc",
    min_delta=0,
    patience=10,
    verbose=1,
    mode="auto",
    restore_best_weights=True,
)

# Khoi tao model
model1 = VGG16(
    weights="imagenet", include_top=False, input_shape=(imageSize, imageSize, 3)
)
optimizer1 = optimizers.Adam()
base_model = model1

# Them cac lop ben tren
x = base_model.output
x = Flatten()(x)
x = Dense(128, activation="relu", name="fc1")(x)
x = Dense(128, activation="relu", name="fc2")(x)
x = Dense(128, activation="relu", name="fc2a")(x)
x = Dense(128, activation="relu", name="fc3")(x)
x = Dropout(0.5)(x)
x = Dense(64, activation="relu", name="fc4")(x)

predictions = Dense(5, activation="softmax")(x)
model = Model(inputs=base_model.input, outputs=predictions)
```

```
# Dong bang cac lop duoi, chi train lop ben tren minh them vao
for layer in base_model.layers:
    layer.trainable = False

model.compile(optimizer="Adam", loss="categorical_crossentropy", metrics=["accuracy
model.fit(
    X_train,
    y_train,
    epochs=50,
    batch_size=64,
    validation_data=(X_test, y_test),
    verbose=1,
    callbacks=[early_stopping, model_checkpoint],
)

# Luu model da train ra file
model.save("models/mymodel.h5")
```

3.7 Code Detection

The gesture detection code supplied below shows how the trained model is integrated into the system. It takes a live video stream from the camera, analyses the visuals to isolate the hand gesture, and then uses the trained model to estimate the meaning of the gesture. Based on the identified motions, the system performs suitable actions, such as changing volume or switching between programs, if the prediction is successful.

The following code excerpt demonstrates the complex process of real-time gesture recognition within this application:

```
import pyautogui
import copy
import cv2
import numpy as np
from keras.models import load_model
import time

# Cac khai bao bien
prediction = ""
score = 0
bgModel = None

gesture_names = {0: "E", 1: "L", 2: "F", 3: "V", 4: "B"}

# Load model tu file da train
model = load_model(
    "C:/Users/bosst/Desktop/Project2/models/mymodel.h5"
)

# Ham de predict xem la ky tu gi
def predict_rgb_image_vgg(image):
    image = np.array(image, dtype="float32")
    image /= 255
    pred_array = model.predict(image)
    print(f"pred_array: {pred_array}")
    result = gesture_names[np.argmax(pred_array)]
    print(f"Result: {result}")
    print(max(pred_array[0]))
    score = float("%0.2f" % (max(pred_array[0]) * 100))
```

```
    print(result)
    return result, score


# Ham xoa nen khoi anh
def remove_background(frame):
    fgmask = bgModel.apply(frame, learningRate=learningRate)
    kernel = np.ones((3, 3), np.uint8)
    fgmask = cv2.erode(fgmask, kernel, iterations=1)
    res = cv2.bitwise_and(frame, frame, mask=fgmask)
    return res


# Khai bao kich thuoc vung detection region
cap_region_x_begin = 0.5
cap_region_y_end = 0.8

# Cac thong so lay threshold
threshold = 60
blurValue = 41
bgSubThreshold = 50  # 50
learningRate = 0

# Nguong du doan ky tu
predThreshold = 95

isBgCaptured = 0  # Bien luu tru da capture background chua

# Camera
camera = cv2.VideoCapture(0)
```

```
camera.set(10, 200)
camera.set(cv2.CAP_PROP_AUTO_EXPOSURE, 0.01)

while camera.isOpened():
    # Doc anh tu webcam
    ret, frame = camera.read()
    # Lam min anh
    frame = cv2.bilateralFilter(frame, 5, 50, 100)
    # Lat ngang anh
    frame = cv2.flip(frame, 1)

    # Ve khung hinh chu nhat vung detection region
    cv2.rectangle(
        frame,
        (int(cap_region_x_begin * frame.shape[1]), 0),
        (frame.shape[1], int(cap_region_y_end * frame.shape[0])),
        (255, 0, 0),
        2,
    )

    # Neu ca capture dc nen
    if isBgCaptured == 1:
        # Tach nen
        img = remove_background(frame)
        cv2.imshow("tach nen", img)
        # Lay vung detection
        img = img[
            0 : int(cap_region_y_end * frame.shape[0]),
            int(cap_region_x_begin * frame.shape[1]) : frame.shape[1],
        ]  # clip the ROI
```

```
# Chuyen ve den trang
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
blur = cv2.GaussianBlur(gray, (blurValue, blurValue), 0)

cv2.imshow("original1", cv2.resize(blur, dsize=None, fx=0.5, fy=0.5))

ret, thresh = cv2.threshold(
    blur, threshold, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU
)

cv2.imshow("Tach hien thi", cv2.resize(thresh, dsize=None, fx=0.5, fy=0.5))

if np.count_nonzero(thresh) / (thresh.shape[0] * thresh.shape[0]) > 0.2:
    # Neu nhu ve duoc hinh ban tay
    if thresh is not None:
        # Dua vao mang de predict
        target = np.stack((thresh,) * 3, axis=-1)
        target = cv2.resize(target, (224, 224))
        target = target.reshape(1, 224, 224, 3)
        prediction, score = predict_rgb_image_vgg(target)

        # Neu probality > nguong du doan thi hien thi
        print(score, prediction)
        if score >= predThreshold:
            cv2.putText(
                frame,
                "Sign:" + prediction,
                (20, 150),
                cv2.FONT_HERSHEY_SIMPLEX,
```

```
                    3,
                    (0, 0, 255),
                    10,
                    lineType=cv2.LINE_AA,
                )
                if prediction == "L":
                    pyautogui.hotkey("alt", "tab")
                elif prediction == "V":
                    pyautogui.press("volumeup")
                elif prediction == "B":
                    pyautogui.press("volumedown")
                elif prediction == "E":
                    pyautogui.press("volumemute")
                    time.sleep(1)

thresh = None

# Xu ly phim bam
k = cv2.waitKey(10)
if k == ord("q"):  # Bam q de thoat
    break
elif k == ord("b"):
    bgModel = cv2.createBackgroundSubtractorMOG2(0, bgSubThreshold)

    isBgCaptured = 1
    cv2.putText(
        frame,
        "Background captured",
        (20, 150),
        cv2.FONT_HERSHEY_SIMPLEX,
```

```
            3,
            (0, 0, 255),
            10,
            lineType=cv2.LINE_AA,
        )
        time.sleep(2)
        print("Background captured")

    elif k == ord("r"):

        bgModel = None
        isBgCaptured = 0
        cv2.putText(
            frame,
            "Background reset",
            (20, 150),
            cv2.FONT_HERSHEY_SIMPLEX,
            3,
            (0, 0, 255),
            10,
            lineType=cv2.LINE_AA,
        )
        print("Background reset")
        time.sleep(1)

    cv2.imshow("original", cv2.resize(frame, dsize=None, fx=0.5, fy=0.5))

cv2.destroyAllWindows()
camera.release()
```

Chapter 4

RESULTS

In this chapter, we present the outcomes and performance of the developed hand gesture recognition system. We showcase the system's ability to recognize hand gestures for controlling various functions and applications. Furthermore, we discuss the limitations encountered during the development process and outline potential avenues for future enhancements.

4.1 Hand Gesture Control Demonstrations

In this section, we provide visual representations of the hand gesture recognition system in action, demonstrating its ability to accurately identify and classify different hand gestures. Each gesture is accompanied by its corresponding action, showcasing the practical application of the system in controlling functions such as media playback, volume adjustment, and application switching.

4.2 Limitations and Future Work

While the developed hand gesture recognition system exhibits promising performance, several limitations were identified during its implementation:

- Challenging Lighting Conditions

  The system's accuracy may be affected under poor lighting conditions or uneven il-

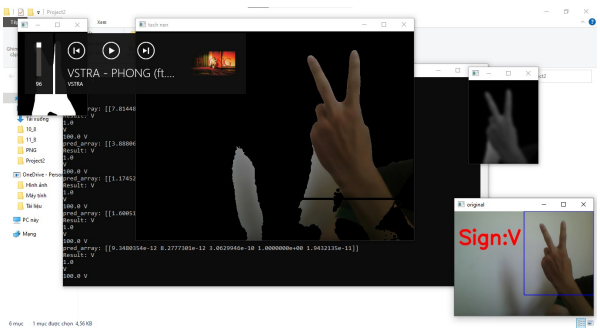

Figure 4.1: Volume Up Gesture

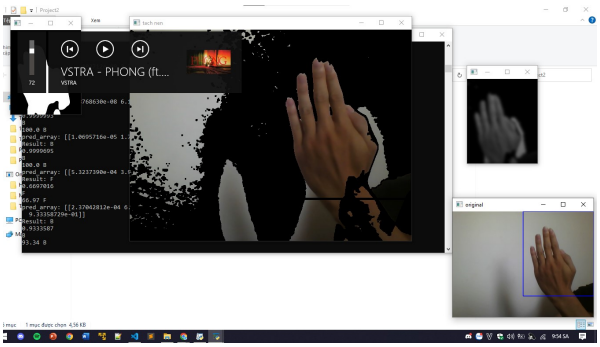

Figure 4.2: Volume Down Gesture

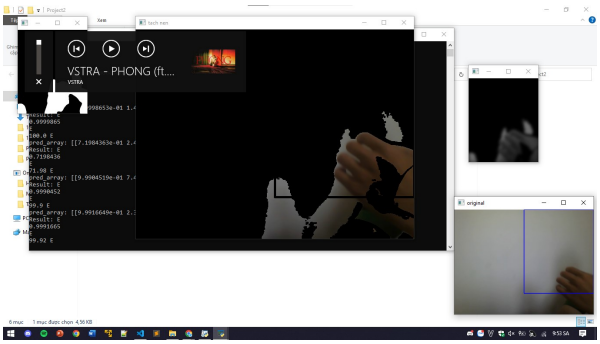

Figure 4.3: Mute Gesture

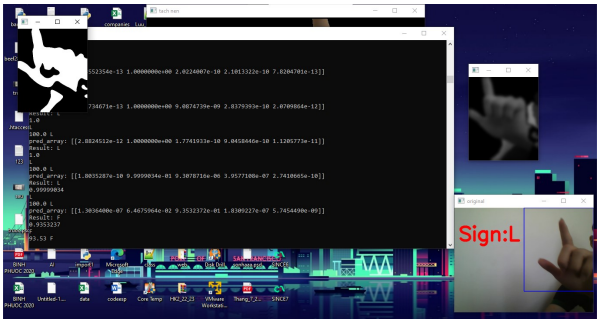

Figure 4.4: Next Track Gesture

lumination. Future work could involve implementing advanced preprocessing techniques to improve performance in various lighting environments.

- Limited Gesture Set

  The current model recognizes a predefined set of hand gestures. Expanding the dataset and incorporating a wider range of gestures would enhance the system's versatility and user interaction possibilities.

4.3 Discussion of Challenges

The development of the hand gesture recognition system posed several challenges:

- Dataset Collection and Labeling

  Gathering a diverse and representative dataset proved challenging due to variations in hand shapes, backgrounds, and lighting conditions. Manual labeling of the dataset required meticulous attention to ensure accurate annotations.

- Model Optimization

Fine-tuning hyperparameters and optimizing the model architecture required iterative experimentation. Achieving a balance between model complexity and generalization capabilities was a key challenge.

4.4 Future Enhancements

To overcome the identified limitations and challenges, several areas of improvement and future enhancements have been identified:

- Dynamic Gesture Recognition Incorporating dynamic hand gestures, such as swipes and rotations, would allow the system to interpret more complex user inputs and perform additional functions.
- Real-time Feedback Implementing real-time visual and auditory feedback for gesture recognition could enhance the user experience and provide confirmation of recognized actions.
- Robustness to Variability Future work could involve developing techniques to improve the system's robustness to variations in hand appearance, orientation, and skin tones.

Chapter 5

CONCLUSION

We review the important results and successes of our work on hand gesture identification for controlling systems and applications in this concluding chapter. We talk about the consequences of our work, emphasize its contributions, and look back on the process of constructing the hand motion detection system. Furthermore, we discuss possible real-world applications and future research directions in the realm of gesture-based interaction.

5.1 Summary of Achievements

We successfully constructed a hand gesture recognition system capable of reliably recognizing and categorizing numerous hand gestures throughout this investigation. Controlling functionality like as media playing, volume adjustment, and application switching were used to show the system's practical applicability. We demonstrated the potential of gesture-based interaction in improving user experiences by using the capabilities of machine learning and computer vision.

5.2 Contributions and Implications

Our work adds to the expanding field of human-computer interaction by allowing users to engage with systems and applications without using their hands. The system's

capacity to identify hand gestures is useful in situations when direct contact with equipment is either impractical or undesirable. This technology has the potential to improve accessibility for those with mobility issues while also revolutionizing how we engage with digital surroundings.

5.3 Reflection

The development of the hand gesture recognition system was both difficult and gratifying. Overcoming technological challenges, fine-tuning models, and dealing with real-world restrictions gave significant insights into the intricacies of developing effective and dependable gesture detection systems. This study demonstrated the value of multi-disciplinary teamwork as well as the iterative nature of research and development.

5.4 Future Directions

While our system has shown outstanding capabilities, there are countless areas for development and extension. Future study might focus on improving the accuracy of gesture detection, investigating unique hand motions, and integrating the system into a variety of applications such as smart homes, augmented reality, and healthcare.

5.5 Final Thoughts

Finally, our research demonstrates the potential of hand gesture identification as a valuable tool for human-computer interaction. The system's creation, successful demonstrations, and ramifications for different sectors underline the technology's importance. As we near the end of our trip, we anticipate seeing the emergence of gesture-based interaction and its transformational influence on how we engage with technology.

We hope that our work serves as a springboard for future developments and stimulates academics, developers, and innovators to continue investigating the potential of gesture-based interaction.

Bibliography

[1] Yann LeCun, Yoshua Bengio, và Geoffrey Hinton, "Deep learning," Nature, vol. 521, no. 7553, pp. 436-444, 2015.

[2] Karen Simonyan và Andrew Zisserman, "Very deep convolutional networks for large-scale image recognition," arXiv preprint arXiv:1409.1556, 2014.

[3] Gary Bradski và Adrian Kaehler, "Learning OpenCV: Computer vision with the OpenCV library," O'Reilly Media, Inc., 2008.

[4] AutoPy Contributors, "AutoPy: A simple, cross-platform GUI automation library for Python," `https://github.com/autopilot-rs/autopy`, 2021.

[5] Al Bewley, "PyAutoGUI: Cross-platform GUI automation for human beings," `https://pyautogui.readthedocs.io/en/latest/`, 2021.

[6] François Chollet, "Keras," `https://keras.io/`, 2015.

[7] "OpenCV Tutorials," `https://docs.opencv.org/4.5.2/d9/df8/tutorial_root.html`, 2021.

[8] H. A. R. Murthy, "A survey on hand gesture recognition," International Journal of Advanced Science and Technology, vol. 28, no. 15, pp. 196-204, 2019.

[9] V. M. Kumar, "Real-time hand gesture recognition using neural networks for human-computer interaction," in 2016 IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT), pp. 680-684, IEEE, 2016.