UNIVERSITY OF MADRAS

GUINDY CAMPUS, CHENNAI- 600 085.

DEPARTMENT OF COMPUTER SCIENCE

ARTIFICAL INTELLIGENCE

TEAM PROJECT

TOPIC

IMAGE CLASSIFICATION USING CNN IN ESP 32

Submitted by

(I M.Sc. Computer Science)

AKSHAYAPRIYAPJ	**36823101**
CHARUMATHI	**36823102**
SWATHYB	**36823107**

Under the Guidance of

Dr. M. SORNAM,Professor,

DEPARTMENT OF COMPUTER SCIENCE

UNIVERSITY OF MADRAS

GUINDY CAMPUS, CHENNAI- 600 085.

DEPARTMENT OF COMPUTER SCIENCE

BONAFIDE CERTIFICATE

This is to certify that the Document work entitled as **"IMAGE CLASSIFICATION USING CNN IN ESP 32"** submitted to the University Of Madras, Chennai in partial fulfillment of the requirement for the award of the degree MASTER OF SCIENCE IN COMPUTER SCIENCE is a Bonafied record of the Mini Project carried out the work has been successfully completed by our team during the period of JANUARY 2024 - APRIL 2024 .

STAFF - IN – CHARGE **HEAD OF THE DEPARTMENT**

Dr.M.SORNAM Dr.S.GOPINATHAN

ACKNOWLEDGEMENT

The acknowledgement of this project is given to all the people who have helped us in completing it.

We express our sincere and profound gratitude to our Head of the Department **Dr. S.GOPINATHAN** for his encouragement and support to do our project.

A special thanks to our guide **. M.SORNAM** who gave us the needed information and encouragement for the successful completion of this project.

We take this opportunity to thank all the staff members and the Lab

Administrators of the Computer Science Department who rendered their help directly to finish our project in time.

We would like to express our hearty thanks to our parents, without whom we would not have come to this level in our life. Our hearty thanks to our friends and well-wishers who supported and encouraged us to complete this project successfully.

TABLE OF CONTENTS

I. PROJECT OBJECTIVE

To classify images with Convolutional Neural Network Algorithm. Build a Convolutional neural network model to classify different categories such as animals, vehicles, or household items using ESP 32 CAMERA .

ABSTRACT

This project aims to develop a Convolutional Neural Network (CNN) model for classifying images into different categories such as animals, vehicles, or household items. The implementation involves using an ESP32 camera for image capture and an ESP8266 module for CNN classification. The model is trained on the YOLOv3 dataset, which is a widely used dataset for object detection and classification tasks. The ESP32 camera captures an image, which is then processed by the ESP8266 module using the trained CNN model to determine the category of the object in the image. The system provides a low-cost and efficient solution for real-time image classification applications.

INTRODUCTION

Image classification is a fundamental task in computer vision with applications ranging from autonomous driving to surveillance systems and augmented reality. Convolutional Neural Networks (CNNs) have emerged as powerful tools for image classification due to their ability to automatically learn features from raw pixel data. In recent years, there has been a growing interest in deploying CNN models on embedded systems for real-time image classification tasks. This project focuses on building a CNN model for classifying images into different categories such as animals, vehicles, or household items. The implementation utilizes the ESP32 camera module for image capture and the ESP8266 module for CNN classification. This combination offers a cost-effective and efficient solution for real-time image classification applications. The CNN model is trained using the YOLOv3 dataset, which provides a diverse range of images annotated with multiple object categories. YOLOv3 is known for its robustness and accuracy in object detection and classification tasks, making it suitable for training the proposed model. By integrating the ESP32 camera for image capture and the ESP8266 module for CNN classification, this project aims to demonstrate a practical and accessible approach to deploying deep learning models on embedded systems. The resulting system has the potential for various applications, including smart surveillance, object recognition in robotics, and assistive technologies for visually impaired individuals.

II.SYSTEM OVERVIEW

The proposed system comprises two main components: the image capture module using ESP32 camera and the image classification module using ESP8266 with a Convolutional Neural Network (CNN) model trained on the YOLOv3 dataset.

1. **Image Capture Module (ESP32 Camera):**

The ESP32 camera module serves as the image capture component of the system. It captures images of the surroundings or objects of interest using its built-in camera sensor. The ESP32 microcontroller processes these images and prepares them for classification.

2. **Image Classification Module (ESP8266 with CNN):**

The ESP8266 module, coupled with the CNN model, performs image classification tasks. The CNN model is pre-trained on the YOLOv3 dataset, which contains a wide variety of annotated images across different categories such as animals, vehicles, and household items. The ESP8266 receives the processed image data from the ESP32 camera module and performs inference using the CNN model to classify the objects present in the image.

3. **Communication Interface:**

The ESP32 camera and ESP8266 modules communicate with each other to facilitate the flow of image data from the capture module to the classification module. This communication can be established using wireless protocols such as Wi-Fi or Bluetooth, depending on the specific implementation requirements.

4. **Deployment and Application:**

Once the image classification process is completed, the system can provide real-time feedback or action based on the identified objects in the image. This feedback can be in the form of notifications, alerts, or control signals, depending on the application scenario. Possible applications include smart surveillance systems, object recognition in robotics, and assistive technologies for visually impaired individuals.

Overall, the system offers a compact, low-cost, and efficient solution for real-time image classification tasks using embedded systems. It leverages the power of deep learning and the convenience of wireless communication to enable a wide range of practical applications in various domains.

TERMINOLOGIES

ABBREVIATIONS	FULL FORM
CNN	
YOLO	
ESP	
ESP 32 CAM	
ESP8266 MOD	
MOD	
FDTI	
AI	
USB	
SD	
OV2640	
GPIO	
RAM	
YOLOv3	
HOG	
RCNN’s	

III.SYSTEM STUDY

Before embarking on the development of the image classification system using ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset, it is essential to conduct a thorough system study to understand the requirements, constraints, and feasibility of the project. The system study involves the following aspects:

1. **Requirements Analysis:**

Identified the specific requirements of the image classification system, including the types of objects to be classified, the desired accuracy and speed of classification, and any constraints related to cost, power consumption, or size. Determined the communication requirements between the ESP32 camera module and the ESP8266 module, such as the data transfer rate and the range of the wireless communication.

2. **Feasibility Study:**

Assessed the technical feasibility of implementing the image classification system using the selected hardware components (ESP32 and ESP8266) and software tools (CNN model trained on YOLOv3 dataset). Evaluated the economic feasibility of the project by estimating the costs associated with hardware components, software development, and any additional resources required.

3. **Risk Analysis:**

Identified potential risks that may arise during the development and deployment of the image classification system, such as technical challenges, compatibility issues, or environmental factors. Developed a risk mitigation plan to address these risks and ensure the successful implementation of the project.

4. **System Design:**

Defined the overall architecture of the image classification system, including the interaction between the ESP32 camera module and the ESP8266 module, as well as the deployment of the CNN model for image classification. Created a detailed design document outlining the components, interfaces, and functionality of the system.

5. **Implementation Plan:**

Developed a timeline for the implementation of the image classification system, including milestones and deliverables.

EXISTING SYSTEM

Before proceeding with the development of a new image classification system using the ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset, it's important to review existing systems or solutions that address similar problems. This allows for a better understanding of current technologies, potential challenges, and areas for improvement. Here's an overview of existing systems in the field of image classification:

1. **Cloud-Based Solutions:**

Many image classification systems rely on cloud-based services such as Amazon Web Services (AWS) Rekognition, Google Cloud Vision API, or Microsoft Azure Computer Vision. These services offer pre-trained models for image classification, making it easy to integrate classification capabilities into applications. However, they may suffer from latency issues and require an internet connection for operation.

2. **Embedded Systems with CNN Models:**

- Some existing systems leverage embedded platforms like Raspberry Pi or Nvidia Jetson for image classification tasks. These systems typically deploy pre-trained CNN models such as MobileNet or ResNet on the embedded hardware. While these solutions offer offline operation and real-time performance, they may require significant computational resources and power consumption.

3. **Custom-Built Solutions:**

- There are also custom-built image classification systems developed using microcontrollers or single-board computers. These systems often involve training CNN models on custom datasets or adapting pre-trained models for specific applications. While they offer flexibility and control over the hardware and software components, they may require expertise in machine learning and embedded systems development.

4. **Hybrid Approaches:**

- Some systems combine cloud-based processing with edge computing for image classification tasks. They offload intensive computations to the cloud for training or batch processing while performing inference on edge devices for real-time applications. This hybrid approach balances the benefits of cloud-based resources with the low latency and privacy advantages of edge computing.

By studying existing systems, the development team can gain insights into the strengths and weaknesses of different approaches to image classification. This knowledge can inform the design and implementation of

the new system using ESP32 camera and ESP8266, helping to address challenges and optimize performance for specific use cases.

PROPOSED SYSTEM

This project proposes a system leveraging Convolutional Neural Networks (CNNs) for real-time image classification on resource-constrained embedded systems. The setup integrates an ESP32 camera module for image acquisition and an ESP8266 module for CNN-based classification using YOLOv3 dataset. The CNN model is trained to discern various categories like animals, vehicles, or household items. The workflow involves capturing images using the ESP32 camera, preprocessing them, and feeding them into the CNN model hosted on the ESP8266 module for rapid classification. This approach illustrates the viability of deploying sophisticated machine learning algorithms on compact, low-power devices for practical applications.

By utilizing the ESP32's image processing capabilities and the ESP8266's computational power, the system offers a cost-effective and efficient solution for on-device image analysis tasks. The project contributes to expanding the capabilities of embedded systems, enabling them to perform complex AI-driven tasks autonomously without relying on external computational resources. The successful implementation of this system opens up possibilities for various real-world applications, including smart surveillance, automated quality control, and intelligent IoT devices.

GOALS OF THE PROJECT

The goal of this project is to demonstrate the feasibility of deploying machine learning algorithms, specifically CNNs, on compact embedded devices for practical applications. By leveraging the capabilities of the ESP32 and ESP8266 modules, the system aims to achieve real-time classification of images into predefined categories such as animals, vehicles, or household items. This introduction sets the stage for exploring the challenges and opportunities associated with deploying CNNs on embedded systems and highlights the significance of the proposed approach in enabling intelligent image analysis directly on edge devices.

IV.SYSTEM REQUIREMENTS

HARDWARE REQUIREMENTS

- Processor: Intel CORE i5 series
- Processor Speed: 1.80 GHz
- RAM: 4 GB RAM
- HDD: 500 GB and above
- Resolution: 1920*108

SOFTWARE REQUIREMENTS

- Operating system: Windows 10
- Language: PYTHON
- Software:
    - Python Idle 3.8,
    - Ardunio Idle,
    - Visual Studio,
    - Flux Ai,
    - Altium 365.

SENSOR REQUIREMENTS

- ESP 32 CAM
- ESP 8266 MOD
- FDTI
- JUMPER
- 5V VOLTAGE
- FDTI ADAPTER
- SERAIL TO USB ADAPTER

V. SENSOR INFORMATION

ESP-32 CAMERA

The ESP32-CAM is a small-sized camera module based on the ESP32-S chip. It allows for capturing images, video streaming, and facial recognition applications. The module includes a camera sensor, an SD card slot to store images or video, and Wi-Fi and Bluetooth connectivity. It's commonly used in IoT projects, security systems, and other applications requiring a compact camera solution with wireless connectivity. In our project ESP-32 Sensor is used to detect images or videos.

Here are some key features of the ESP32-CAM:

- ESP32 Microcontroller: The board is built around the ESP32, a powerful and versatile microcontroller that supports both Wi-Fi and Bluetooth connectivity. It has a dual-core processor, ample RAM, and various peripherals.
- Camera Module: The ESP32-CAM integrates a small camera module, typically an OV2640 or OV7670, capable of capturing images and video. The camera can be used to capture still images or stream video to a host device.
- GPIO Pins: The ESP32-CAM features a set of general-purpose input/output (GPIO) pins that allow you to connect additional sensors, actuators, or other components to expand the functionality of your project.
- Storage Options: The board offers different storage options for storing images and other data. It includes a microSD card slot for external storage, as well as built-in flash memory for storing firmware and other files.
- Programming: The ESP32-CAM can be programmed using the Arduino IDE, which provides a user-friendly development environment for writing and uploading code. There are also alternative programming options available, such as MicroPython or the Espressif IDF (IoT Development Framework).
- The ESP32-CAM is commonly used in applications such as surveillance systems, home automation, robotics, and IoT projects that require image capture and wireless connectivity. Its compact size and affordable price make it a popular choice for hobbyists and developers.

ESP8266MOD

The ESP8266MOD is a Wi-Fi module based on the ESP8266 chip, which is widely used for Internet of Things (IoT) applications. It integrates a microcontroller unit with Wi-Fi capabilities, allowing devices to connect to Wi-Fi networks and communicate with other devices or servers over the internet. The ESP8266MOD module typically comes in a small form factor with a set of GPIO pins for interfacing with sensors, actuators, and other components. It's popular among hobbyists and professionals alike for its low cost, ease of use, and versatility in IoT projects. In our project ESP 8266 MOD is used to implement CNN algorithm and classify the images for both live or stored images.

FDTI

FTDI chips are widely used for USB-to-serial communication and are commonly found in microcontroller-based projects where a USB connection is required for programming or debugging. FTDI chips are popular because they are easy to use and offer reliable communication between a microcontroller and a computer. They come in various forms, including standalone chips, USB-to-serial converter cables, and modules that integrate USB functionality directly into a circuit board. FTDI's chips and modules have become ubiquitous in the maker and electronics hobbyist communities due to their widespread compatibility and ease of integration. They are often used with microcontrollers like Arduino, ESP8266, and ESP32 for serial communication and programming purposes.

VI. ALGORITHM SPECIFICATION

CONVULATIONAL NEURAL NETWORKS ALGORITHM

A Convolutional Neural Network (CNN) is a type of deep learning algorithm specifically designed for image processing and recognition tasks. Compared to alternative classification models, CNNs require less preprocessing as they can automatically learn hierarchical feature representations from raw input images. They excel at assigning importance to various objects and features within the images through convolutional layers, which apply filters to detect local patterns.

The connectivity pattern in CNNs is inspired by the visual cortex in the human brain, where neurons respond to specific regions or receptive fields in the visual space. This architecture enables CNNs to effectively capture spatial relationships and patterns in images. By stacking multiple convolutional and pooling layers, CNNs can learn increasingly complex features, leading to high accuracy in tasks like image classification, object detection, and segmentation

SCOPE OF CONVULATIONAL NEURAL NETWORKS:-

The scope of Convolutional Neural Networks (CNNs) is quite vast. CNNs are primarily used for image recognition and computer vision tasks. Some of the key applications and areas where CNNs are widely used include:

- **Image Classification**: CNNs excel in classifying images into various categories or classes. They can accurately identify objects, animals, people, and various other visual elements in images.
- **Object Detection**: CNNs can locate and identify multiple objects within an image. They are used in applications such as self-driving car technology, security surveillance, and augmented reality.
- **Facial Recognition**: CNNs are effective in recognizing and verifying faces, enabling facial recognition systems used for access control, surveillance, and user identification.
- **Natural Language Processing**: CNNs can be used in text classification tasks, such as sentiment analysis, spam detection, and speech recognition.
- **Medical Imaging**: CNNs are used to analyze medical images, assisting in the detection and diagnosis of various diseases, such as cancer, tumors, and abnormalities.
- **Autonomous Vehicles**: CNNs are crucial in enabling object detection, lane detection, and traffic sign recognition in autonomous vehicles.
- **Video Analysis**: CNNs are utilized in video understanding tasks, including action recognition, video classification, and video surveillance.
- **Style Transfer**: CNNs can transfer the style of one image to another, creating artistic and visually appealing effects.These are just a few examples of the scope of CNNs. With ongoing advancements in deep learning, CNNs are continuously finding new applications and possibilities in various fields.

ADVANTAGES OF CONVULATIONAL NEURAL NETWORKS:

- **Automatic Feature Extraction**: CNNs have the ability to automatically learn and extract features from images without requiring manual feature engineering. This makes them highly effective in tasks such as object detection and image classification.
- **Spatial Hierarchies**: CNNs are designed to recognize patterns and objects in a hierarchical manner. They can learn low-level features such as edges and corners, and then gradually combine them to form higher-level features and representations. This allows the network to capture spatial relationships and context in images.
- **Translation Invariance**: CNNs are capable of detecting the same features in different parts of an image, regardless of their position. This translation invariance property makes CNNs robust to variations in object placement or orientation, improving their generalization performance.
- **Parameter Sharing**: CNNs use parameter sharing to dramatically reduce the number of parameters and the computational complexity of the model. By sharing weights across different parts of the image, CNNs can efficiently learn and detect features in a wide range of spatial locations.

DISADVANTAGES OF CONVULATIONAL NEURAL NETWORKS:

- **High Computational Requirements**: CNNs can be computationally expensive, especially when dealing with large datasets or complex architectures. Training and inference on CNNs may require powerful hardware and significant computational resources.
- **Overfitting**: Like any deep learning model, CNNs are prone to overfitting, where the model becomes highly specialized to the training data and performs poorly on unseen data. Regularization techniques, such as dropout and weight decay, can help mitigate this issue.
- **Interpretability:** CNNs are often considered as black box models, making it challenging to interpret or understand the decision-making process. It can be difficult to determine why a CNN is making certain predictions, which can be a limitation in some domains where interpretability is crucial.
- **Data Requirements**: CNNs typically require a large amount of labeled training data to achieve high performance. Acquiring and annotating such datasets can be time-consuming and expensive, especially for specialized domains with limited data availability.

VII. DATASET SPECIFICATION

YOLOv3

YOLOv3, short for "You Only Look Once version 3," is a state-of-the-art deep learning algorithm for real-time object detection in images and videos. Developed by Joseph Redmon and Ali Farhadi at the University of Washington, YOLOv3 builds upon its predecessors' successes, addressing limitations and improving performance. With its remarkable speed and accuracy, YOLOv3 has become a cornerstone in the field of computer vision and is widely used in various applications, including surveillance, autonomous vehicles, and robotics. At the core of YOLOv3 is the concept of one-stage object detection. Unlike traditional two-stage detectors, which separate object localization and classification into distinct steps, YOLOv3 performs both tasks simultaneously in a single neural network. This unified approach not only simplifies the architecture but also significantly reduces inference time, enabling real-time performance on resource-constrained devices.

The key idea behind YOLO v3 is to divide an input image into a grid and predict bounding boxes and class probabilities directly on the grid cells. Instead of sliding a window or using a region proposal network, YOLO v3 performs detection in a single pass. This makes it extremely fast and efficient compared to other object detection algorithms.

YOLO v3 uses a deep convolutional neural network (CNN) to process the input image and predict the bounding boxes and class probabilities. The network architecture comprises several convolutional layers, which are subsequently followed by fully connected layers. It also incorporates skip connections, which allow information from earlier layers to be used in later layers, enhancing the detection performance.

One of the significant improvements in YOLO v3 is the introduction of multiple detection scales. It applies detection at three different scales to detect objects of varying sizes in the image. This multi-scale approach helps improve detection accuracy, particularly for small objects.

YOLO v3 is capable of detecting and localizing multiple objects within an image in real time. It has been widely used in various applications, including autonomous vehicles, surveillance systems, and video analysis.

ADVANTAGES OF YOLOv3 DATASET

YOLOv3 offers several advantages that contribute to its popularity and widespread adoption in the field of computer vision and object detection:

1. **Real-Time Performance:** YOLOv3 is capable of achieving real-time object detection, even on resource-constrained devices such as embedded systems and smartphones. Its efficient architecture and unified approach

to object detection enable high-speed inference, making it suitable for applications requiring fast response times, such as surveillance and autonomous vehicles.

2. **Unified Architecture:** Unlike traditional two-stage object detection algorithms that separate object localization and classification into distinct stages, YOLOv3 performs both tasks simultaneously within a single neural network. This unified architecture simplifies the model and reduces computational overhead, leading to faster inference and lower memory requirements.

3. **High Accuracy:** Despite its real-time performance, YOLOv3 achieves high accuracy in object detection tasks. By incorporating advanced features such as multi-scale prediction, feature pyramid networks, and anchor-based bounding box prediction, YOLOv3 can accurately localize and classify objects of various sizes, shapes, and orientations.

4. **Multi-Class Detection:** YOLOv3 is capable of detecting multiple object classes within a single image or video frame. It can identify and localize objects belonging to different categories simultaneously, making it suitable for applications where multiple objects of interest may be present.

5. **Versatility:** YOLOv3 is a versatile algorithm that can be applied to a wide range of object detection tasks across diverse domains. Whether it's detecting pedestrians on a sidewalk, vehicles on a road, or household objects in a room, YOLOv3 can adapt to different scenarios and environments with high accuracy and efficiency.

6. **Ease of Use:** YOLOv3 is relatively easy to implement and deploy, thanks to its open-source nature and comprehensive documentation. Pre-trained models and software libraries are available, simplifying the process of integrating YOLOv3 into custom applications and projects. Additionally, the algorithm's straightforward architecture and training pipeline make it accessible to both researchers and developers.

7. **Community Support:** YOLOv3 benefits from a large and active community of developers, researchers, and enthusiasts who contribute to its development, improvement, and optimization. This vibrant ecosystem ensures ongoing support, updates, and enhancements to the algorithm, keeping it at the forefront of object detection research and applications.

VIII. MODULES

The image classification system using ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset consists of several key modules:

1. **ESP32 Camera Module:**
   - This module includes an ESP32 microcontroller and a camera sensor. It is responsible for capturing images of the surroundings or objects of interest.

2. **Image Preprocessing Module:**
   This module preprocesses the captured images to enhance quality or reduce noise. It may include resizing, normalization, or color correction of the images.

3. **ESP8266 Module:**
   This module includes an ESP8266 microcontroller. It receives the processed image data from the ESP32 camera module.

4. **Wireless Communication Module:**
   This module enables wireless communication between the ESP32 camera module and the ESP8266 module. It uses protocols such as Wi-Fi or Bluetooth for data transmission.

5. **CNN Model Module:**
   - This module contains the CNN model trained on the YOLOv3 dataset.
   - It performs inference on the processed image data to classify objects in the images.

6. **Classification Result Module:**
   This module outputs the classification results, indicating the detected objects and their corresponding categories. It provides feedback or action based on the classification results.

7. **Integration Module:**
   - This module integrates the functionalities of the ESP32 camera module, ESP8266 module, image preprocessing module, wireless communication module, CNN model module, classification result module, and feedback/action module. It ensures seamless operation and interaction between the different modules of the system. These modules work together to enable the image classification system to capture, process, classify, and act upon images in real-time, providing a comprehensive solution for various applications such as object recognition, surveillance, and assistive technologies.

IX.WORKING

The working of the proposed image classification system using the ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset involves several steps:

1. **Image Capture:**
   - The ESP32 camera module captures images of the surroundings or objects of interest using its built-in camera sensor. These images are typically in the form of raw pixel data.

2. **Image Preprocessing:**
   - The captured images may undergo preprocessing to enhance quality or reduce noise. This preprocessing step can include resizing, normalization, or color correction, depending on the requirements of the CNN model.

3. **Wireless Communication:**
   - The ESP32 camera module communicates wirelessly with the ESP8266 module to transmit the processed image data. This communication can be established using Wi-Fi or another wireless protocol supported by both modules.

4. **Image Classification:**
   - The ESP8266 module receives the processed image data from the ESP32 camera module and performs inference using the pre-trained CNN model. The CNN model is trained on the YOLOv3 dataset and is capable of classifying objects into predefined categories such as animals, vehicles, or household items.

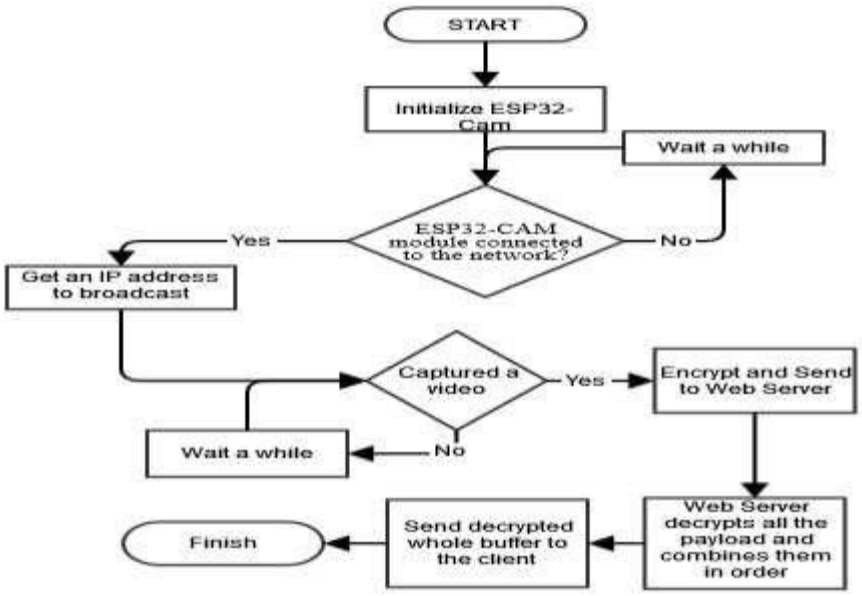
5. **Classification Result:**
   - Once the inference is completed, the ESP8266 module outputs the classification results, indicating the detected objects and their corresponding categories. This information can be sent to a display, stored in memory, or transmitted to another device for further processing.

6. **Feedback or Action:**
   - Based on the classification results, the system may provide real-time feedback or take action accordingly. For example, in a smart surveillance system, the system may trigger an alert if a suspicious object or person is detected.

7. Continuous Operation:
   - The system operates continuously, capturing images, performing inference, and providing feedback or action as required. This real-time operation enables applications such as object recognition, monitoring, or automation.

Overall, the working of the image classification system involves capturing images using the ESP32 camera, transmitting image data wirelessly to the ESP8266 module, performing inference using the CNN model, and providing feedback or action based on the classification results. This integrated approach offers a cost-effective and efficient solution for real-time image classification applications using embedded systems.

FLOWCHART



DETECTION

In the context of image processing and computer vision, "detection" refers to the process of identifying and locating objects within an image. Detection is a fundamental task in various applications, including object recognition, surveillance, autonomous driving, and augmented reality. Techniques for detection vary depending on the specific requirements of the application, the complexity of the scenes, and the available computational resources. Common approaches to detection include traditional computer vision methods, such as Haar cascades and HOG (Histogram of Oriented Gradients), as well as deep learning-based approaches, such as Convolutional Neural Networks (CNNs) and Region-based Convolutional Neural Networks (R-CNNs).

It involves determining the presence, location, and extent of objects of interest in the image. Here's how the detection process typically works:

1. **Object Localization & Classification:**

Detection often begins with localizing objects within the image by identifying their bounding boxes. This step involves determining the coordinates of rectangles or bounding boxes that enclose the detected objects. Once the objects are localized, the system proceeds to classify them into predefined categories or classes. This classification step involves determining what each detected object represents, such as a person, car, or animal.

2. **Confidence Scores:**

Detection algorithms often provide confidence scores or probabilities associated with each detected object and its corresponding class. These scores indicate the level of certainty or confidence that the algorithm has in its detections.

3. **Post-processing:**

After detection and classification, post-processing techniques may be applied to refine the results or improve accuracy. This could include filtering out low-confidence detections, refining bounding box coordinates, or performing additional analysis on the detected objects.

CLASSIFICATION

Classification is the process of assigning predefined labels or categories to objects based on their characteristics or features. In the context of image processing and computer vision, image classification involves categorizing images into distinct classes or categories. This task is fundamental in various applications such as object recognition, scene understanding, medical imaging, and quality control. Here's how the classification process typically works:

1. **Feature Extraction:**

The first step in image classification involves extracting relevant features from the input image. These features could include color histograms, texture patterns, edge orientations, or deep features learned by convolutional neural networks (CNNs).

2. **Model Training:**

Once the features are extracted, a classification model is trained using a labeled dataset. The dataset consists of input images along with their corresponding labels or categories. During training, the model learns to associate specific features with each class and adjust its parameters to minimize classification errors.

3. **Inference:**

After training, the classification model is used to predict the category of new, unseen images. This process, known as inference, involves feeding the extracted features of the input image into the trained model and obtaining the model's predictions.

4. **Decision Making:**

Based on the model's predictions, a decision is made regarding the category or label assigned to the input image. The classification result indicates which class the input image belongs to, providing valuable information for downstream tasks or applications.

5. **Evaluation:**

The performance of the classification model is evaluated using metrics such as accuracy, precision, recall, and F1 score. These metrics measure the model's ability to correctly classify images across different classes and provide insights into its effectiveness and reliability.

PREDICTION

In the context of machine learning and artificial intelligence, "prediction" refers to the process of estimating or forecasting a future outcome or value based on historical data or patterns learned from a training dataset. Predictions are made by trained models using input features or data points, and they aim to provide insights or make informed decisions about future events. Here's how the prediction process typically works:

1. **Model Training:**

Prediction begins with training a machine learning model using historical data that contains input features and corresponding target values or labels. During training, the model learns patterns and relationships in the data, adjusting its parameters to minimize prediction errors.

2.**Feature Extraction:**

When making predictions, input features or data points are extracted from the new or unseen instances. These features represent relevant information that the model uses to make predictions. Feature extraction may involve preprocessing, normalization, or transformation of the input data.

3. **Inference:**

The trained model performs inference on the input features to generate predictions. This involves applying the learned patterns and relationships to the input data to estimate the target values or outcomes. In classification tasks, predictions typically correspond to class labels, while in regression tasks, predictions correspond to continuous numerical values.

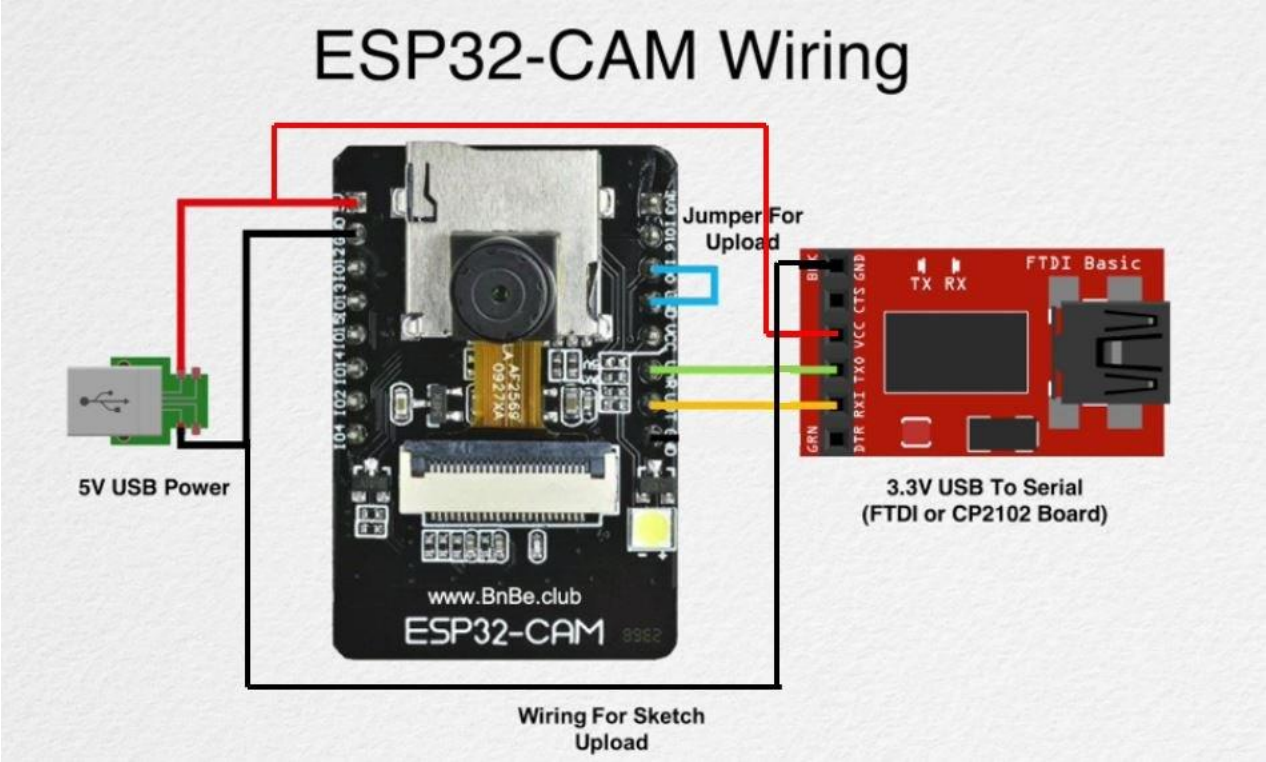
4. **Evaluation and Validation:**

The quality and accuracy of predictions are evaluated using validation datasets or metrics such as accuracy, precision, recall, mean squared error, or R-squared. These metrics assess the model's ability to generalize to new, unseen data and provide insights into its performance.

5. **Deployment and Monitoring:**

Once validated, the trained model can be deployed into production environments to make real-time predictions on new data. Continuous monitoring and evaluation of the model's performance are essential to ensure its accuracy and reliability over time.

CONNECTION OF ESP 32CAM



X.CODING

ESP32 Cam Live Video Streaming In Python Opencv:

```
#include <WebServer.h>
#include <WiFi.h>
#include <esp32cam.h>
```

```
const char* WIFI_SSID = "Fawad";
const char* WIFI_PASS = "computer007";
WebServer server(80);
static auto loRes = esp32cam::Resolution::find(320, 240);
static auto midRes = esp32cam::Resolution::find(350, 530);
static auto hiRes = esp32cam::Resolution::find(800, 600);
void serveJpg()
{
  auto frame = esp32cam::capture();
  if (frame == nullptr) {
    Serial.println("CAPTURE FAIL");
    server.send(503, "", "");
    return;
  }
  Serial.printf("CAPTURE OK %dx%d %db\n", frame->getWidth(), frame->getHeight(),
                static_cast<int>(frame->size()));

  server.setContentLength(frame->size());
  server.send(200, "image/jpeg");
  WiFiClient client = server.client();
  frame->writeTo(client);
}
void handleJpgLo()
{
  if (!esp32cam::Camera.changeResolution(loRes)) {
    Serial.println("SET-LO-RES FAIL");
  }
  serveJpg();
}

void handleJpgHi()
{
  if (!esp32cam::Camera.changeResolution(hiRes)) {
    Serial.println("SET-HI-RES FAIL");
  }
  serveJpg();
```

```
}
void handleJpgMid()
{
  if (!esp32cam::Camera.changeResolution(midRes)) {
    Serial.println("SET-MID-RES FAIL");
  }
  serveJpg();
}
void  setup(){
  Serial.begin(115200);
  Serial.println();
  {
    using namespace esp32cam;
    Config cfg;
    cfg.setPins(pins::AiThinker);
    cfg.setResolution(hiRes);
    cfg.setBufferCount(2);
    cfg.setJpeg(80);
    bool ok = Camera.begin(cfg);
    Serial.println(ok ? "CAMERA OK" : "CAMERA FAIL");
  }
  WiFi.persistent(false);
  WiFi.mode(WIFI_STA);
  WiFi.begin(WIFI_SSID, WIFI_PASS);
  while (WiFi.status() != WL_CONNECTED) {
    delay(500);
  }
  Serial.print("http://");
  Serial.println(WiFi.localIP());
  Serial.println("  /cam-lo.jpg");
  Serial.println("  /cam-hi.jpg");
  Serial.println("  /cam-mid.jpg");
  server.on("/cam-lo.jpg", handleJpgLo);
  server.on("/cam-hi.jpg", handleJpgHi);
  server.on("/cam-mid.jpg", handleJpgMid);
  server.begin();
```

```
}
void loop()
{
server.handleClient();
}
```

PYTHON CODE FOR VIDEO STREAMING USING ESP32 CAM:

```
import cv2
import urllib.request
import numpy as np
# Replace the URL with the IP camera's stream URL
url = 'http://192.168.43.219/cam-hi.jpg'
cv2.namedWindow("live Cam Testing", cv2.WINDOW_AUTOSIZE)
# Create a VideoCapture object
cap = cv2.VideoCapture(url)
# Check if the IP camera stream is opened successfully
if not cap.isOpened():
    print("Failed to open the IP camera stream")
    exit()
# Read and display video frames
while True:
    # Read a frame from the video stream
    img_resp=urllib.request.urlopen(url)
    imgnp=np.array(bytearray(img_resp.read()),dtype=np.uint8)
    #ret, frame = cap.read()
    im = cv2.imdecode(imgnp,-1)
    cv2.imshow('live Cam Testing',im)
    key=cv2.waitKey(5)
    if key==ord('q'):
        break
cap.release()
cv2.destroyAllWindows()
```

PYTHON OPENCV YOLO V3 CODE :

```
import cv2
import numpy as np
```

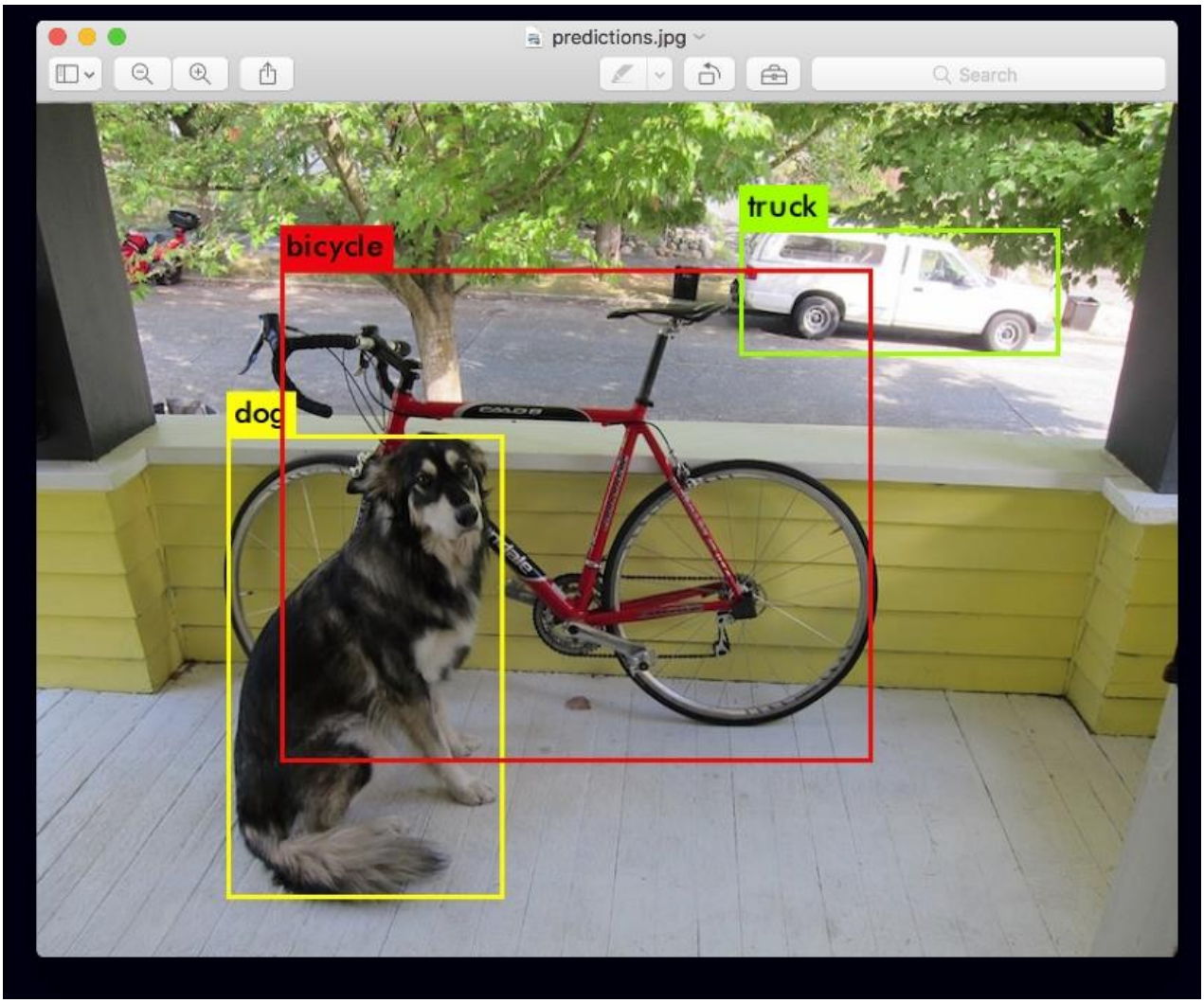
```
import urllib.request
url = 'http://192.168.43.219/cam-hi.jpg”
cap = cv2.VideoCapture(url)
whT=320
confThreshold = 0.5
nmsThreshold = 0.3
classesfile='coco.names'
classNames=[]
with open(classesfile,'rt') as f:
    classNames=f.read().rstrip('\n').split('\n')
modelConfig = 'yolov3.cfg'
modelWeights= 'yolov3.weights'
net = cv2.dnn.readNetFromDarknet(modelConfig,modelWeights)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
def findObject(outputs,im):
    hT,wT,cT = im.shape
    bbox = []
    classIds = []
    confs = []
    found_cat = False
    found_bird = False
    for output in outputs:
            classId = np.argmax(scores)
            confidence = scores[classId]
            if confidence > confThreshold:
                w,h = int(det[2]*wT), int(det[3]*hT)
                x,y = int((det[0]*wT)-w/2), int((det[1]*hT)-h/2)
                bbox.append([x,y,w,h])
                classIds.append(classId)
                confs.append(float(confidence))
    indices = cv2.dnn.NMSBoxes(bbox,confs,confThreshold,nmsThreshold)
    print(indices)
    for i in indices:
        i = i[0]
        box = bbox[i]
```

```
        x,y,w,h = box[0],box[1],box[2],box[3]
        if classNames[classIds[i]] == 'bird':
            found_bird = True
        elif classNames[classIds[i]] == 'cat':
            found_cat = True
        cv2.rectangle(im,(x,y),(x+w,y+h),(255,0,255),2)
        cv2.putText(im, f'{classNames[classIds[i]].upper()} {int(confs[i]*100)}%', (x,y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,0,255), 2)
while True:
    img_resp=urllib.request.urlopen(url)
    imgnp=np.array(bytearray(img_resp.read()),dtype=np.uint8)
    im = cv2.imdecode(imgnp,-1)
    sucess, img= cap.read()
    blob=cv2.dnn.blobFromImage(im,1/255,(whT,whT),[0,0,0],1,crop=False)
    net.setInput(blob)
    layernames=net.getLayerNames()
    outputNames = [layernames[i[0]-1] for i in net.getUnconnectedOutLayers()]
    outputs = net.forward(outputNames)
    findObject(outputs,im)
    cv2.imshow('IMage',im)
    cv2.waitKey(1)
import cv2
import numpy as np
import urllib.request
url = 'http://192.168.43.219/cam-hi.jpg'
cap = cv2.VideoCapture(url)
whT=320
confThreshold = 0.5
nmsThreshold = 0.3
classesfile='coco.names'
classNames=[]
with open(classesfile,'rt') as f:
    classNames=f.read().rstrip('\n').split('\n')
#print(classNames)
modelConfig = 'yolov3.cfg'
modelWeights= 'yolov3.weights'
```

```
net = cv2.dnn.readNetFromDarknet(modelConfig,modelWeights)
net.setPreferableBackend(cv2.dnn.DNN_BACKEND_OPENCV)
net.setPreferableTarget(cv2.dnn.DNN_TARGET_CPU)
def findObject(outputs,im):
    hT,wT,cT = im.shape
    bbox = []
    classIds = []
    confs = []
    found_cat = False
    found_bird = False
    for output in outputs:
        for det in output:
            scores = det[5:]
            classId = np.argmax(scores)
            confidence = scores[classId]
            if confidence > confThreshold:
                w,h = int(det[2]*wT), int(det[3]*hT)
                x,y = int((det[0]*wT)-w/2), int((det[1]*hT)-h/2)
                bbox.append([x,y,w,h])
                classIds.append(classId)
                confs.append(float(confidence))
    #print(len(bbox))
    indices = cv2.dnn.NMSBoxes(bbox,confs,confThreshold,nmsThreshold)
    print(indices)
    for i in indices:
        i = i[0]
        box = bbox[i]
        x,y,w,h = box[0],box[1],box[2],box[3]
        if classNames[classIds[i]] == 'bird':
            found_bird = True
        elif classNames[classIds[i]] == 'cat':
            found_cat = True
        if classNames[classIds[i]]=='bird':
            cv2.rectangle(im,(x,y),(x+w,y+h),(255,0,255),2)
            cv2.putText(im, f'{classNames[classIds[i]].upper()} {int(confs[i]*100)}%', (x,y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,0,255), 2)
```

```
            print('bird')
            print(found_bird)
        if classNames[classIds[i]]=='cat':
 cv2.rectangle(im,(x,y),(x+w,y+h),(255,0,255),2)
            cv2.putText(im, f'{classNames[classIds[i]].upper()} {int(confs[i]*100)}%', (x,y-10),
cv2.FONT_HERSHEY_SIMPLEX, 0.6, (255,0,255), 2)
            print('cat')
            print(found_cat)
        if found_cat and found_bird:
            print('alert')
while True:
    img_resp=urllib.request.urlopen(url)
    imgnp=np.array(bytearray(img_resp.read()),dtype=np.uint8)
    im = cv2.imdecode(imgnp,-1)
    sucess, img= cap.read()
    blob=cv2.dnn.blobFromImage(im,1/255,(whT,whT),[0,0,0],1,crop=False)
    net.setInput(blob)
    layernames=net.getLayerNames()
    #print(layernames)
    outputNames = [layernames[i[0]-1] for i in net.getUnconnectedOutLayers()]
    #print(net.getUnconnectedOutLayers())
    outputs = net.forward(outputNames)
    #print(outputs[0].shape)
    #print(outputs[1].shape)
    #print(outputs[2].shape)
    #print(outputs[0][0])
    findObject(outputs,im)
    cv2.imshow('IMage',im)
    cv2.waitKey(1)
```

XI.OUTPUT


predictions.jpg
Search
truck
bicycle
dog

XII.FUTURE SCOPE

The proposed image classification system using ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset has significant future scope. One avenue is enhancing the system's classification accuracy and speed through optimization techniques, such as model compression, quantization, and hardware acceleration. Additionally, integrating advanced features like real-time object tracking, multi-object detection, and scene understanding can further enhance its capabilities. Moreover, exploring transfer learning techniques to adapt the pre-trained CNN model to new environments or tasks can improve its versatility and applicability. Furthermore, expanding the system's dataset and class labels can enable it to classify a broader range of objects and scenes accurately. Overall, continuous research and development efforts can enhance the system's performance, expand its application areas, and make it a more robust and versatile solution for real-time image classification tasks in various domains.

XIII.ADVANTAGES

The image classification system using ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset offers several advantages:

1. **Real-time Classification:** The system provides real-time image classification, enabling quick and efficient detection and recognition of objects in captured images.

2. **Low-cost Solution:** By leveraging affordable hardware components such as ESP32 and ESP8266 modules, the system offers a cost-effective solution for image classification applications, making it accessible to a wide range of users.

3. **Embedded Deployment:** The system can be deployed on embedded platforms, allowing for compact and lightweight implementations suitable for IoT devices, robotics, and edge computing applications.

4. **Wireless Connectivity**: Utilizing wireless communication between the ESP32 camera and ESP8266 module enables flexible deployment and operation without the need for physical connections, enhancing convenience and scalability.

5. **Deep Learning Capabilities:** By using a CNN model trained on the YOLOv3 dataset, the system benefits from deep learning techniques for accurate and robust image classification across various object categories.

6. **Versatility:** The system can be adapted and customized for different applications and environments by training the CNN model on specific datasets or fine-tuning model parameters.

7. **Potential for Expansion:** With continuous research and development, the system has the potential to incorporate additional features, improve classification accuracy, and expand its capabilities to address evolving requirements and challenges in image classification tasks.

XIV.CONCLUSION

In conclusion, the image classification system utilizing ESP32 camera and ESP8266 with a CNN model trained on the YOLOv3 dataset presents a promising solution with vast potential. Through ongoing research and development efforts, this system can evolve to meet the demands of diverse applications. Optimization techniques can enhance its accuracy and speed, while the integration of advanced features like real-time tracking and scene understanding can broaden its utility. Moreover, exploring transfer learning methods and expanding the dataset can further improve its performance and adaptability to new environments. As technology continues to advance, this system holds the promise of becoming a versatile and robust solution for real-time image classification across various domains. With continuous innovation and refinement, it can address emerging challenges and empower applications ranging from surveillance and robotics to assistive technologies and beyond.

XV. REFERENCES

- https://www.kdnuggets.com/2022/05/image-classification-convolutional-neural-networks-cnns.html
- https://datagen.tech/guides/image-classification/image-classification-using-cnn/

- https://www.tensorflow.org/tutorials/images/cnn
- https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4
- https://towardsdatascience.com/using-convolutional-neural-network-for-image-classification-5997bfd0ede4
- https://fluxai.com
- YOLO: Real-Time Object Detection (pjreddie.com)