CHAPTER 1

INTRODUCTION

In recent years, advancements in artificial intelligence (AI) and machine learning (ML) have transformed the way we interact with technology, particularly in the realm of image recognition. Image classification, a key application of computer vision, enables machines to analyze and interpret visual data, making it possible to identify objects, scenes, and activities within images. This capability has numerous applications, including in healthcare, security, retail, and autonomous vehicles.

This project focuses on developing a web application that utilizes a pre-trained deep learning model, MobileNetV2, for image classification tasks. MobileNetV2 is a lightweight convolutional neural network (CNN) designed for mobile and edge devices, allowing for efficient processing with reduced computational resources while maintaining high accuracy.

It is particularly well-suited for real-time applications where speed and efficiency are critical. The web application is built using Flask, a popular micro web framework for Python. It provides a user-friendly interface that allows users to upload images directly from their devices. Upon receiving an image, the application processes it and feeds it into the MobileNetV2 model, which predicts the most likely category for the image along with a confidence score.

This interaction between the user and the model demonstrates the practical implementation of deep learning in a web environment..

The primary objectives of this mini-project are:

1 **Implement a Flask web application** that facilitates image uploads and displays classification results.
2 **Utilize the MobileNetV2 model** for efficient image classification, leveraging its capabilities to provide accurate predictions.
3 **Provide an intuitive user experience**, making advanced image recognition technology accessible to users without requiring technical expertise.

3.1 Significance of the Study

This study focuses on the development and implementation of a web-based image classification application using Flask and a pre-trained deep learning model, MobileNetV2. The application allows users to upload images and receive predictions about the contents of those images, which can be utilized in various domains such as nutrition, health monitoring, and educational purposes.Scope of the Study

- **Health Enthusiasts**: Individuals interested in monitoring their dietary intake through image classification of food items.
- **Students and Educators**: Users in educational settings who wish to explore machine learning and image processing concepts.
- **Developers and Researchers**: Professionals looking to understand and build upon existing machine learning frameworks and applications

- **Model Accuracy**: The predictions are based on the capabilities of the MobileNetV2 model, which may not be 100% accurate for all image types.
- **Image Quality**: The quality and resolution of uploaded images can significantly impact the prediction results.

Scope of Classification: The application is limited to the classes recognized by the ImageNet

.

3.2 Scope of the Study

Th The scope of the study for the provided Flask application code focuses on implementing an image classification system using a pre-trained MobileNetV2 model. This scope involves various technical aspects, including web application development, image processing, and machine learning integration. Below is a detailed scope of the stud **.**

Web Application Development

- **Framework Selection**: The study will involve the use of the Flask framework for building a lightweight web application. Flask is chosen for its simplicity and ease of use, making it suitable for deploying machine learning models.

- **Routing and User Interaction**: The application will handle HTTP requests through Flask's routing system. Users will interact with the application primarily through a file upload form, where they can submit images for classification.

- **Template Rendering**: The application will utilize HTML templates (e.g., upload.html and result.html) to render the user interface, allowing users to upload images and view the classification results.

2. Machine Learning Integration

- **Model Selection**: The study will focus on integrating a pre-trained MobileNetV2 model, which is widely used for image classification tasks due to its efficiency and accuracy.
- **Prediction Pipeline**: The scope includes developing a pipeline for loading and preprocessing images, feeding them into the MobileNetV2 model, and decoding the predictions into human-readable labels.
- **Result Interpretation**: The application will return the top prediction from the model, along with the associated confidence level, allowing users to understand the model's output.

3. Image Processing

- **Preprocessing**: The study will explore image preprocessing techniques, such as resizing and normalizing images to fit the input requirements of the MobileNetV2 model.
- **File Handling**: The application will handle file uploads securely, ensuring that images are saved to a specified directory on the server, and processed without exposing security risks.

4. User Experience and Interface Design

- **User-Friendly Interface**: The application will feature a simple and intuitive user interface, where users can easily upload images and receive predictions.
- **Visualization**: The result page will display the prediction along with the uploaded image, enhancing the user experience by providing visual feedback.

5. System Performance and Optimization

- **Model Performance**: The study will evaluate the performance of the MobileNetV2 model in terms of prediction accuracy and speed within the web application.
- **Server-Side Performance**: The application will be tested for its ability to handle multiple requests, manage file uploads, and return predictions in a timely manner.

6. Scalability and Future Enhancements

- **Scalability**: The study will consider the scalability of the application, including how it can be extended to handle more complex use cases, such as multiple image uploads

CHAPTER 2

LITERATURE SURVEY

Image Classification using Deep Learning Image classification is a fundamental task in computer vision that involves assigning a label or category to an input image. With the rise of deep learning, convolutional neural networks (CNNs) have become the dominant approach for image classification tasks. CNNs are able to automatically learn features from raw image data, making them powerful tools for a wide range of applications. Several studies have surveyed the progress and advancements in image classification using deep learning. Rawat and Wang (2017) provided a comprehensive review of CNN architectures and their applications in computer vision. They discussed the evolution of CNN models from Alex Net to more recent architectures like ResNet and Dense Net, highlighting their key contributions and performance on benchmark datasets et al. (2018) focused on the theoretical aspects of CNN design, analysing the impact of depth, width, and cardinality on model performance. They also reviewed techniques for improving CNN efficiency, such as depth wise separable convolutions and group convolutions, which are particularly relevant for mobile and embedded applications. Lately, there has been growing interest in applying deep learning to remote sensing and satellite imagery. Zhu et al. (2017) provided a survey on deep learning for Earth observation data, covering topics like scene classification, object detection, and change detection. They discussed the unique challenges of remote sensing data, such as high spatial resolution and domain adaptation, and how CNN models can be adapted to address these challenges.

MobileNetV2: Efficient CNN Architecture MobileNetV2 is a lightweight CNN architecture designed for mobile and embedded devices, developed by researchers at Google. It builds upon the success of the original MobileNet model by introducing several key innovations, such as:

1. Inverted residual blocks: MobileNetV2 uses an inverted residual structure, where the input and output of the block are thin bottleneck layers, while the intermediate layers have expanded feature maps. This allows for efficient computation and memory usage.
2. Linear bottlenecks: The paper shows that using a linear activation in the narrow layers of the network is important for performance. This is contrary to the popular belief that non-linearities are always needed.
3. Depthwise separable convolutions: MobileNetV2 heavily relies on depthwise separable convolutions, which factorize a standard convolution into a depthwise convolution and a 1x1 pointwise convolution. This significantly reduces the number of parameters and computations compared to standard convolutions.

Sandler et al. (2018) demonstrated that MobileNetV2 achieves state-of-the-art performance on ImageNet classification while being 1.4x faster and 1.7x more efficient than previous models like MobileNetV1 and ShuffleNet. The authors also showed the versatility of MobileNetV2 by applying it to other tasks like object detection and semantic segmentation.

CHAPTER 3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

- Basic Functionality: The existing system allows users to upload images for classification.
- User Interface: It features a simple web interface with an upload form.
- Image Processing: The system processes images using a pre-trained MobileNetV2 model.
- Prediction Output: It returns the predicted class label and confidence score.
- File Handling: Uploaded images are saved to a static directory on the server.
- Limited Feedback: Users receive minimal feedback beyond the prediction result.
- Single Prediction: The system predicts only the top class for each uploaded image.
- No Image Validation: There is no validation for image types or sizes before processing.
- Error Handling: Basic error handling is implemented for file uploads.
- No User History: There is no tracking of user uploads or previous predictions.
- Static Model: The model is static and does not adapt or improve over time.
- Single Language Support: The interface is likely limited to one language.
- Resource Intensive: The model runs on the server, which may require significant resources.
- No Scalability: The system may struggle with multiple simultaneous users.

3.2 PROPOSED SYSTEM

- Enhanced User Interface: The proposed system features a more interactive and user-friendly web interface.
- Multiple Image Uploads: Users can upload multiple images at once for batch processing.
- Advanced Feedback: The system provides detailed feedback, including multiple predictions per image.
- Image Validation: It validates image types and sizes before processing to improve reliability.
- User Accounts: Users can create accounts to track their upload history and predictions.
- Dynamic Model Updates: The model can be updated periodically to improve accuracy and performance.
- Multi-Language Support: The interface supports multiple languages for broader accessibility.
- Mobile Responsiveness: The web application is designed to be mobile-friendly for easier access.
- Asynchronous Processing: Image processing can be handled asynchronously to improve user experience.
- Visualization Tools: The system includes visualizations of prediction confidence and model performance.
- Community Features: Users can share their predictions and images with a community for feedback.
- API Integration: The system can integrate with external APIs for additional functionalities.
- Scalability: The architecture supports scaling to handle increased user load effectively.
- Enhanced Security: Improved security measures are implemented to protect user data and uploads.
- Educational Resources: The system offers resources to educate users about image classification and model workings

CHAPTER 4
REQUIREMENT ANALYSIS

1.1 HARDWARE REQUIREMENTS:

User Devices

Smartphone or Tablet:

- **Processor:** Dual-core CPU (Intel i5 or equivalent) or higher.
- **RAM:** Minimum 8 GB.
- **Storage:** Minimum 10 GB available space.
- **Operating System:** iOS 11.0 or later, Android 7.0 or later.
- **Display:** HD resolution (1280x720 pixels) or higher.
- **Internet Connectivity:** Wi-Fi and/or mobile data (4G/5G).

Server Infrastructure

Web Server:

- **Processor:** Minimum Quad-core Intel Xeon or equivalent.
- **RAM:** Minimum 16 GB for handling multiple concurrent requests and image processing.
- **Storage:** SSD with a minimum of 500 GB for fast read/write operations, especially for storing model weights and uploaded images.
- **Operating System:** Linux (Ubuntu 18.04 or later) or Windows Server 2016 or later.
- **Network:** Gigabit Ethernet connection for efficient data transfer.
- **Database:** MySQL or PostgreSQL with SSD storage for fast access to any necessary data.

1.2 Software Requirements

- **Frontend:** HTML5, CSS3, JavaScript for creating the user interface.
- **Web Development Tools:** Visual Studio Code or Sublime Text for code editing and development.
- **Version Control:** Git (GitHub, GitLab, or Bitbucket) for source code management and collaboration.
- **APIs for Image Classification:** TensorFlow and Keras for loading the MobileNetV2 model, handling image preprocessing, and making predictions.
- **Image Processing:** Pillow (PIL) for image handling.
- **Browser DevTools:** Chrome or Firefox for frontend debugging and testing.

- **Additional Libraries:**
    - NumPy for numerical operations.
    - Flask for creating the web application.

CHAPTER 5
SYSTEM DESIGN

System Components

The Flask application for image classification using MobileNetV2 consists of the following key components:

User Interface

- **Web Interface**: The application provides a web interface built using HTML, CSS, and JavaScript for users to upload images and view the classification results.
- **Upload Page**: Users can access the upload page to select an image file from their device.
- **Result Page**: After uploading an image, the result page displays the predicted class label, description, and confidence level.

Backend Server

- **Flask Application**: The core of the system is the Flask web application that handles user requests and processes the uploaded images.
- **API Endpoints**: The application exposes API endpoints for handling image uploads and returning the classification results.

Image Processing

- **Image Loading**: The application uses the Pillow (PIL) library to load the uploaded image.
- **Image Preprocessing**: The image is preprocessed using the preprocess_input function from the MobileNetV2 module to match the expected input format of the model.

Machine Learning Model

- **MobileNetV2**: The pre-trained MobileNetV2 model is loaded from the TensorFlow Keras library. The model is used for classifying the uploaded images.

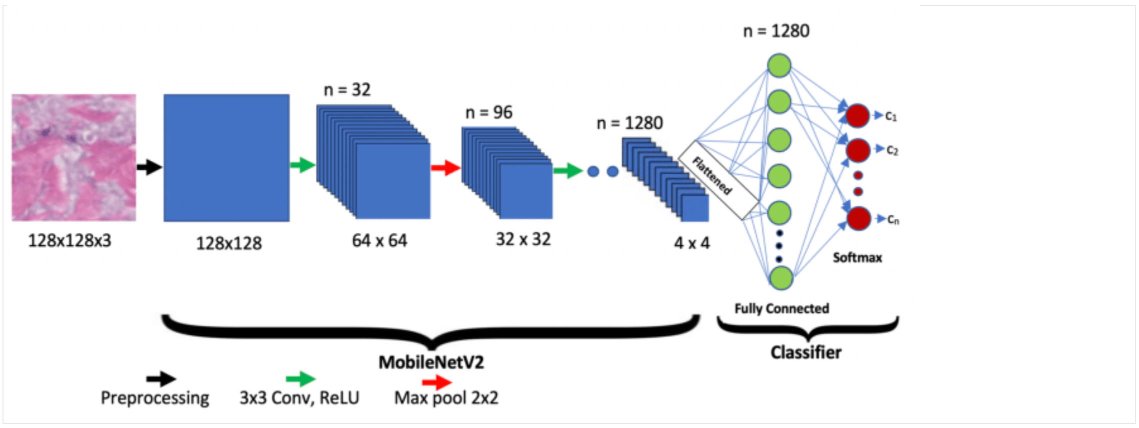


Fig 3.1Architecture of Image Classification System

Description:
The architecture of the Image Classification System involves a client-server model where users upload images through a Flask-based web interface. The server processes these images by loading and pre-processing them to meet the input requirements of the MobileNetV2 model. The pre-trained MobileNetV2 model then predicts the image class, and the results, including the predicted label and confidence score, are returned to the user through a dynamically rendered web page. This architecture efficiently combines web development, image processing, and machine learning.

- **Model Prediction**: The preprocessed image is passed through the loaded MobileNetV2 model to obtain the classification predictions.

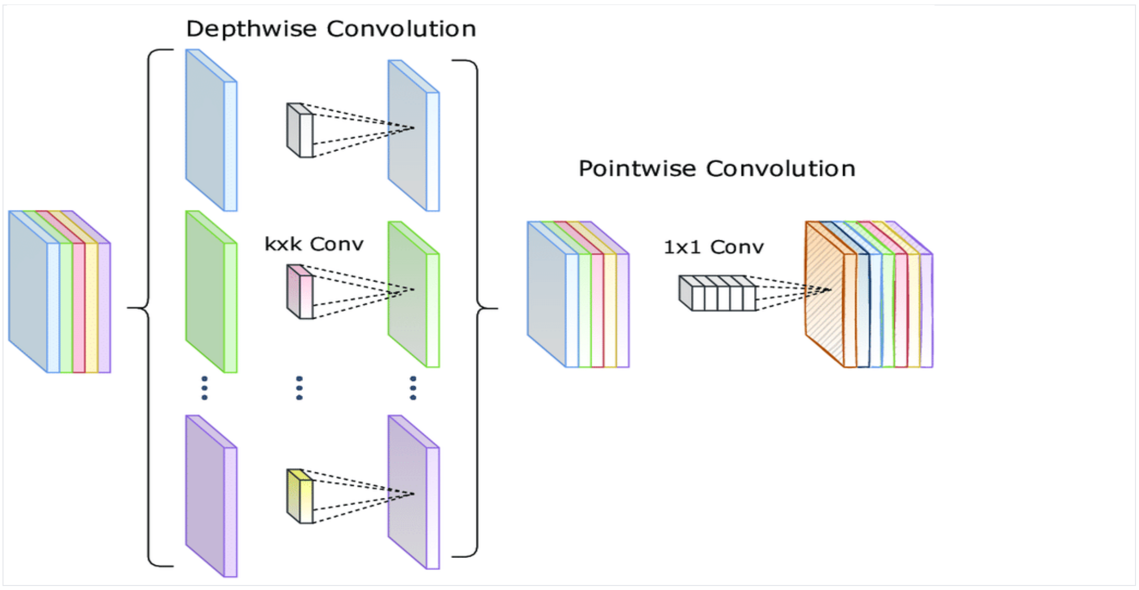


Fig 3.2 Block Diagram of MobileNetV2

Description:

MobileNetV2 is structured around an efficient architecture designed for mobile and edge devices, characterized by its use of **inverted residual blocks**. Each block consists of three layers: a 1x1 convolution for feature expansion, a depthwise convolution for lightweight filtering, and another 1x1 convolution for feature compression without non-linearity. The architecture also incorporates **residual connections** to facilitate better gradient flow during training, enhancing performance while maintaining a low computational footprint. The overall design emphasizes efficiency, making it suitable for applications requiring real-time processing on devices with limited resources

- **Prediction Decoding**: The raw model predictions are decoded using the decode_predictions function from the MobileNetV2 module to obtain the class label, description, and confidence level.

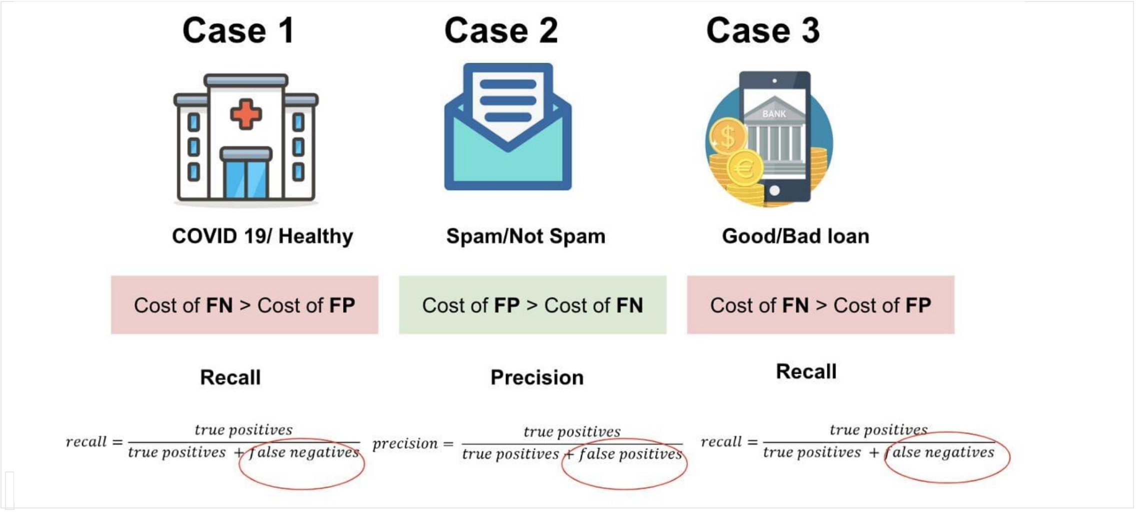

Case 1
Case 2
Case 3
BANK
COVID 19/ Healthy
Spam/Not Spam
Good/Bad loan
Cost of FN > Cost of FP
Cost of FP > Cost of FN
Cost of FN > Cost of FP
Recall
Precision
Recall
recall = true positives / (true positives + false negatives)
precision = true positives / (true positives + false positives)
recall = true positives / (true positives + false negatives)

Fig 3.3 Example of Preprocessed Image

Description:
A preprocessed image is an input image that has been transformed to meet the requirements of a machine learning model. This typically involves resizing the image to a specific dimension, such as 224x224 pixels for MobileNetV2. Additionally, the pixel values are scaled and normalized, often by subtracting the mean and dividing by the standard deviation, to enhance model performance. Finally, the image is converted into a format suitable for batch processing, such as a NumPy array

File Storage

- **Uploads Directory**: The application creates an uploads directory within the static folder to store the uploaded images.
- **Image Saving**: The uploaded image is saved to the uploads directory using the file.save() method.

Database (Not Applicable)

- The provided code does not include any database integration. If required, a database can be added to store user information, uploaded images, or classification history.

Real-Time Processing (Not Applicable)

- The application processes images in real-time upon user upload and does not require any additional real-time processing components.

Notification System (Not Applicable)

- The provided code does not include any notification system. If required, notifications can be added to inform users about successful uploads or provide updates on the classification process

5.1 UML DIAGRAMS

Introduction

- Unified Modelling Language Diagrams
- The unified modelling language allows the software engineer to express an analysis model using the modelling notation that is governed by a set of syntactic semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagrams, which is as follows:

Software Techniques

- This view represents the system from the user's perspective.
- The analysis representation describes a usage scenario from the end-user's perspective.
    - Structural model view
- In this model the data and functionality are arrived from inside the
- This model view models the static structures.
    - Behavioral Model View
    - It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.
    - Implementation Model View
    - In this the structural and behavioral as parts of the system are represented as they are to be built.
    - Environmental Model View

- In this the structural and behavioral aspects of the environment in
- which the system is to be implemented are represented.
- UML is specifically constructed through two different domains they are:
- ✓ UML Analysis modelling, this focuses on the user model and structural model views
- of the system.
- ✓ UML design modelling, which focuses on the behavioral modelling, implementation

- modelling and environmental model views.
- Use case Diagrams represent the functionality of the system from a user's point of view. Use cases are used during requirements elicitation and analysis to represent the functionality of the system. Use cases focus on the behavior of the system from external point of view. Actors are external entities that interact with the system. Examples of actors include users like administrator, bank customer ...etc., or another system like central database.

Unified Modelling Language

The Unified Modelling Language (UML) is a standard language for writing software blueprints. The UML may be used to visualize, specify, construct, and document the artifacts of a software-intensive system. The UML is appropriate for modelling systems ranging from enterprise information systems to distributed Web-based applications and even to hard real time embedded systems. It is a very expressive language, addressing all the views needed to develop and then deploy such systems. The UML is only a language and so is just one part of a software development method. The UML is process independent, although optimally it should be used in a process that is use case driven, architecture-centric, iterative, and incremental.

The UML is a language for
Visualizing.

Specifying.

Constructing.

Documenting.

- The artifacts of a software-intensive system. A modelling language is a language whose vocabulary and rules focus on the conceptual and physical representation of a system. Modelling yields an understanding of a system. The vocabulary of the UML encompasses three kinds of building blocks:
    - Things
    - Relationships
    - UML Diagrams
    - The unified modeling language allows the software engineer to express an analysis model using the modeling notation that is governed by a set of syntactic semantic and pragmatic rules. A UML system is represented using five different views that describe the system from distinctly different perspective. Each view is defined by a set of diagrams, which is as follows:

- User Model View

- This view represents the system from the user’s perspective.
- 2.The analysis representation describes a usage scenario from the end-user’s perspective.
- Structural model view
- In this model the data and functionality are arrived from inside the system.
- 2.This model view models the static structures.

- Behavioral Model View

- It represents the dynamic of behavioral as parts of the system, depicting the interactions of collection between various structural elements described in the user model and structural model view.
- Implementation Model View
- In this the structural and behavioral as parts of the system are represented as they are to be built.
- Environmental Model View
- In these the structural and behavioral aspects of the environment in which the system is to be implemented are represented.
- UML is specifically constructed through two different domains they are:
- 1.UML Analysis modeling, this focuses on the user model and structural model views of the system.
- 2.UML design modeling, which focuses on the behavioral modeling, implementation modeling and environmental model views.

5.1.1 Use Case Diagram:

The use case diagram identifies and describes the different interactions a user can have with the system. For the Integrated Wellness and Nutrition Monitoring System, this involves entering food items, displaying nutritional values, and calculating BMI. By visualizing these interactions, the diagram helps ensure that all essential user needs and functionalities are considered during the design phase.

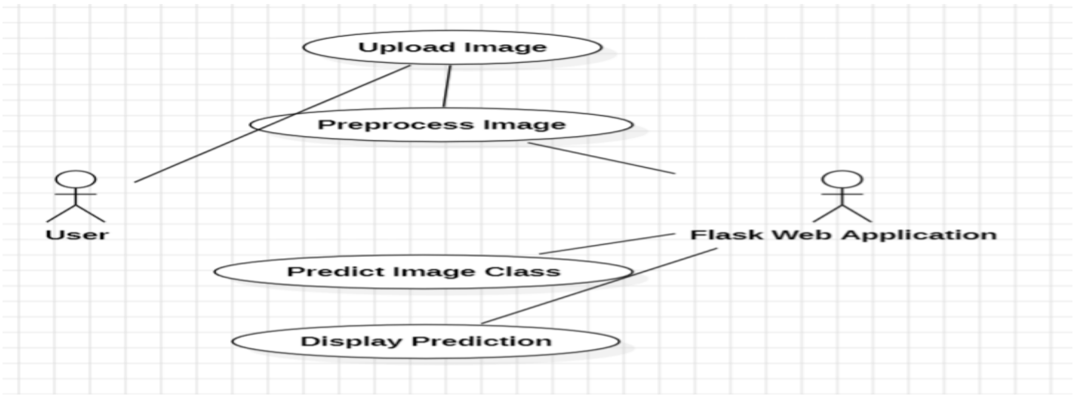


Fig 5.1 Use Case Diagram

5.1.2 Sequence Diagram:

The sequence diagram for the Integrated Wellness and Nutrition Monitoring System helps in visualizing the detailed interactions between users and system components over time. It guides development, supports testing, helps in error handling, and facilitates clear communication among stakeholders. By mapping out these interactions, the diagram ensures that the system behaves as expected and meets user needs.

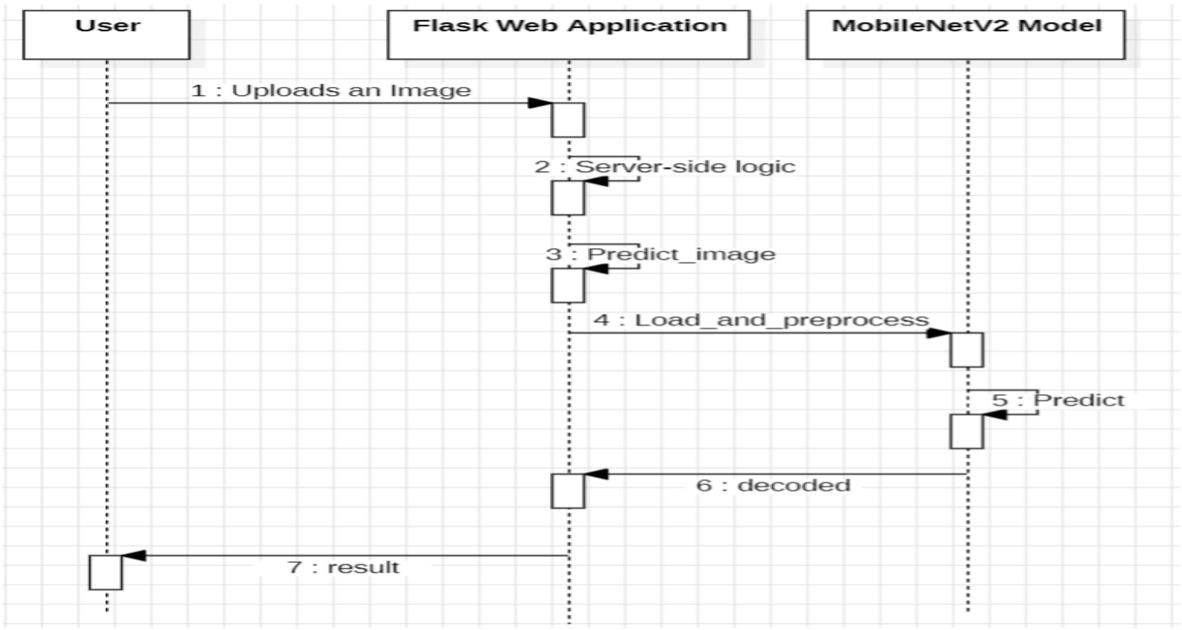


Fig 5.2 Sequence Diagram

5.1.3 Activity Diagram:

The activity diagram for the Integrated Wellness and Nutrition Monitoring System helps in visualizing and understanding the workflow and processes involved. It aids in defining and optimizing processes, identifying decision points, and improving communication among stakeholders. By providing a clear depiction of the sequence of activities and their interactions, the diagram supports development, testing, and overall system design

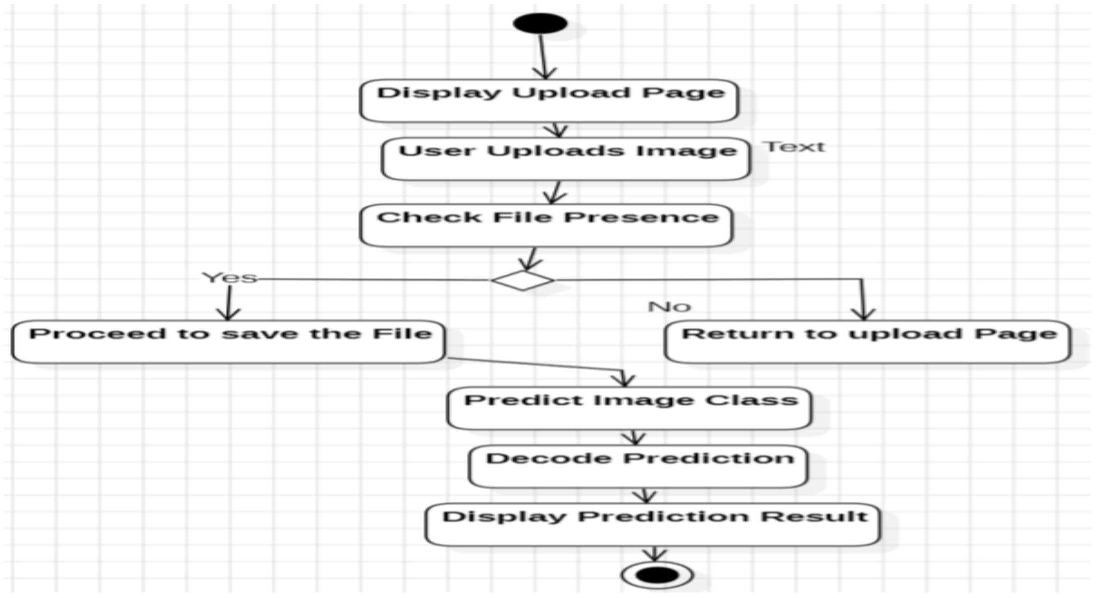


Fig 5.3 Activity Diagram

CHAPTER 6
IMPLEMENTATION

6.1 PROBLEM STATEMENT

Creating an "MobileNetV2 integration for high-performance image classification web app " The objective of this project is to develop a web-based application using Flask that allows users to upload images for classification. The application utilizes a pre-trained MobileNetV2 model to predict the contents of the uploaded images. Users will receive feedback on the predicted class of the image along with the confidence level of the prediction..

6.2 IMPLEMENTATION STEPS

1. Define Objectives and Requirements

Image Upload and Processing:

- The application should allow users to upload images through a web interface. It must handle various file types (e.g., JPEG, PNG) and ensure that uploaded images are saved in a designated directory for processing.

Image Classification Using Pre-trained Model:

- The application must utilize the pre-trained MobileNetV2 model to classify the uploaded images. It should preprocess the images to the required input format, perform predictions, and return the predicted class label along with a confidence score.

User-Friendly Result Display:

- After processing the image, the application should render a results page that displays the prediction results, including the predicted class, a human-readable label, and the confidence percentage, ensuring a clear and informative user experience.

2. Research and Feasibility Study

- Technology Review: The application utilizes the pre-trained MobileNetV2 model, which is known for its efficiency in image classification tasks, particularly in mobile and web applications. Researching similar models and their performance metrics can help in understanding the strengths and limitations of MobileNetV2, especially in terms of accuracy and speed for real-time predictions.

- Prototype Feasibility: The feasibility of integrating image processing capabilities with Flask needs to be assessed. This includes evaluating the efficiency of loading and preprocessing images using Keras utilities, as well as the model's ability to handle various image formats and sizes. Testing the end-to-end workflow from image upload to prediction will help identify potential bottlenecks.
- Regulations and Standards: Since the application handles user-uploaded images, it is important to research relevant data privacy regulations (e.g., GDPR, CCPA) to ensure compliance. Understanding how to securely store and manage user data, including implementing proper consent mechanisms, is critical for building user trust.

3. Design and Development

- **Software Design:**
  - Frontend Development: The Flask application provides a simple user interface for uploading images. The upload.html template is rendered when the user visits the root URL (/), allowing them to select a file for upload.
  - When the user submits the form, the upload_file function in the Flask application handles the request. It checks if a file was uploaded and saves it to the static/uploads directory using the os module.
  - After saving the uploaded image, the predict_image function is called to process the image using the pre-trained MobileNetV2 model. The predicted class, label, and confidence score are returned and passed to the result.html template for display.
- **Hardware Integration:**
  - No Hardware Integration: The provided code does not involve any hardware integration. It is a software-based image classification application that runs on a Flask server.
  - Potential Hardware Integration: If the application needs to integrate with hardware devices, such as cameras or scanners, additional code would be required to handle the communication and data transfer between the Flask application and the hardware components.
  - Scalability and Compatibility: The Flask application's architecture should be designed to accommodate potential hardware integration in the future, ensuring scalability and compatibility with various devices that may be added to the system.

4. Prototype and Development

- The code loads the pre-trained MobileNetV2 model with ImageNet weights using MobileNetV2(weights='imagenet')
- The load_and_preprocess_image function loads an image from the specified path, resizes it to the required size (224x224), and preprocesses it using preprocess_input from tensorflow.keras.applications.mobilenet
- The Flask application is created using Flask(__name__

5. Testing and Validation

- Lab Testing: Test the prototype in a controlled environment to ensure accurate predicting values.
- Field Testing: Conduct real-world tests with actual users to validate the system's performance, accuracy, and user experience.
- User Feedback: Gather feedback from potential users to identify strengths and areas for improvement, making necessary adjustments to the system.

6. Iterate and Improve

- Refine Design: Based on testing results and user feedback, enhance both the hardware (if applicable) and software design to improve functionality and usability.
- Address Issues: Resolve any identified issues, such as inaccuracies in data processing or user interface inefficiencies, ensuring a robust and reliable system.

7. Compliance and Certification

- Health and Data Privacy Standards: Ensure the system complies with relevant health standards, such as HIPAA for data privacy and security, and nutritional guidelines.
- Certification: Obtain necessary certifications for the software, particularly for any components that may require medical or health-related approvals.

8. Manufacturing and Production (if applicable)

- Manufacturing Plan: Develop a plan for the production of any physical components or devices associated with the system, ensuring scalability and cost-effectiveness.
- Quality Control: Implement stringent quality control measures to guarantee the reliability and consistency of the system, especially for devices that integrate with the software.

9. Marketing and User Education

- Market Strategy: Develop a strategy for marketing the system to target users, including predicting values individuals, enthusiasts, and those with specific classification needs.
- Educational Resources: Provide comprehensive resources, including tutorials and guides, to educate users on how to effectively utilize the system's features for optimal prediction management.

6.2 SOURCE CODE :

app.py

```
from flask import Flask, request, render_template
import os
from tensorflow.keras.applications import MobileNetV2
from tensorflow.keras.preprocessing import image
from tensorflow.keras.applications.mobilenet_v2 import preprocess_input, decode_predictions
import numpy as np

app = Flask(__name__)

# Load the pre-trained MobileNetV2 model
model = MobileNetV2(weights='imagenet')

# Function to load and preprocess image
def load_and_preprocess_image(img_path):
    img = image.load_img(img_path, target_size=(224, 224))
    img_array = image.img_to_array(img)
    img_array = np.expand_dims(img_array, axis=0)
    return preprocess_input(img_array)

# Function to predict and return the prediction
def predict_image(img_path):
    img_array = load_and_preprocess_image(img_path)
    predictions = model.predict(img_array)
    decoded_predictions = decode_predictions(predictions, top=1)[0][0]
    confidence = round(decoded_predictions[2] * 100)  # Round to the nearest whole number
    return (decoded_predictions[0], decoded_predictions[1], confidence)

@app.route('/', methods=['GET', 'POST'])
def upload_file():
    if request.method == 'POST':
        if 'file' not in request.files:
            return 'No file part'
        file = request.files['file']
        if file.filename == '':
            return 'No selected file'
        if file:
            # Ensure the uploads directory exists
            uploads_dir = os.path.join('static', 'uploads')
            os.makedirs(uploads_dir, exist_ok=True)  # Create the directory if it doesn't exist
            # Save the file to the uploads directory
            img_path = os.path.join(uploads_dir, file.filename)
            file.save(img_path)
            prediction = predict_image(img_path)
            return render_template('result.html', prediction=prediction, image_filename=file.filename)
    return render_template('upload.html')

if __name__ == '__main__':
    app.run(debug=True)
```

result.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Prediction Result</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            color: #333;
            margin: 0;
            padding: 20px;
            display: flex;
            flex-direction: column;
            align-items: center;
        }
        h1 {
            color: #4CAF50;
        }
        .result-container {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            padding: 20px;
            text-align: center;
            width: 300px;
            margin-top: 20px;
        }
        p {
            font-size: 18px;
            margin: 10px 0;
        }
        a {
            display: inline-block;
            margin-top: 20px;
            padding: 10px 20px;
            background-color: #4CAF50;
            color: white;
            text-decoration: none;
            border-radius: 5px;
            transition: background-color 0.3s;
        }
        a:hover {
            background-color: #45a049;
        }
        img {
            max-width: 100%;
            border-radius: 8px;
            margin-top: 20px;
        }
    </style>
</head>
<body>
    <h1>Prediction Result</h1>
    <div class="result-container">
        <p>Predicted Label: <strong>{{ prediction[1] }}</strong></p>
        <p>Confidence: <strong>{{ prediction[2] }}%</strong></p>
        <img src="{{ url_for('static', filename='uploads/' + image_filename) }}" alt="Uploaded Image">
    </div>
    <a href="/">Upload another image</a>
</body>
</html>
```

Upload.html

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Image Upload</title>
    <style>
        body {
            font-family: Arial, sans-serif;
            background-color: #f4f4f4;
            color: #333;
            margin: 0;
            padding: 20px;
            display: flex;
            flex-direction: column;
            align-items: center;
        }
        h1 {
            color: #4CAF50;
            margin-bottom: 20px;
        }
        .upload-container {
            background-color: white;
            border-radius: 8px;
            box-shadow: 0 2px 10px rgba(0, 0, 0, 0.1);
            padding: 30px;
            text-align: center;
            width: 300px;
            margin-top: 20px;
        }
        input[type="file"] {
            margin-bottom: 20px;
            padding: 10px;
            border: 1px solid #ccc;
            border-radius: 5px;
            width: 100%;
            box-sizing: border-box;
        }
        input[type="submit"] {
            padding: 10px 20px;
            background-color: #4CAF50;
            color: white;
            border: none;
            border-radius: 5px;
            cursor: pointer;
            transition: background-color 0.3s;
        }
        input[type="submit"]:hover {
            background-color: #45a049;
        }
    </style>
</head>
<body>
    <h1>Upload an Image for Prediction</h1>
    <div class="upload-container">
        <form method="POST" enctype="multipart/form-data">
            <input type="file" name="file" accept="image/*" required>
            <input type="submit" value="Upload">
        </form>
    </div>
</body>
</html>
```

MobileNetV2 integration for high-performance image classification web app Users Project Directory Structure :

```
image_classification_app/
│
├── app.py                  # Main application file
│
├── requirements.txt           # List of dependencies
│
├── static/                 # Directory for static files
│   ├── uploads/              # Directory for uploaded images
│   ├── styles.css            # CSS file for styling (optional)
│   └── images/               # Directory for any static images (optional)
│
├── templates/              # Directory for HTML templates
│   ├── upload.html           # Template for the image upload form
│   └── result.html           # Template for displaying prediction results
│
└── model/                  # Directory for model-related files (optional)
    └── mobilenetv2.h5        # Saved MobileNetV2 model file (if applicable)
```

d

Project Directory Structure

CHAPTER 7
SCREENSHOTS

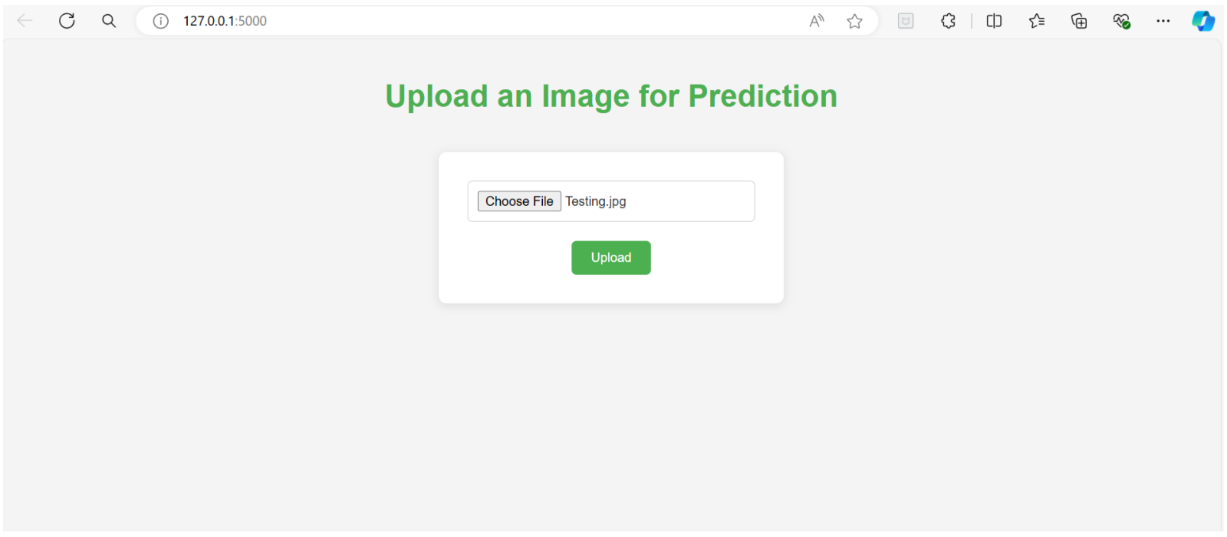
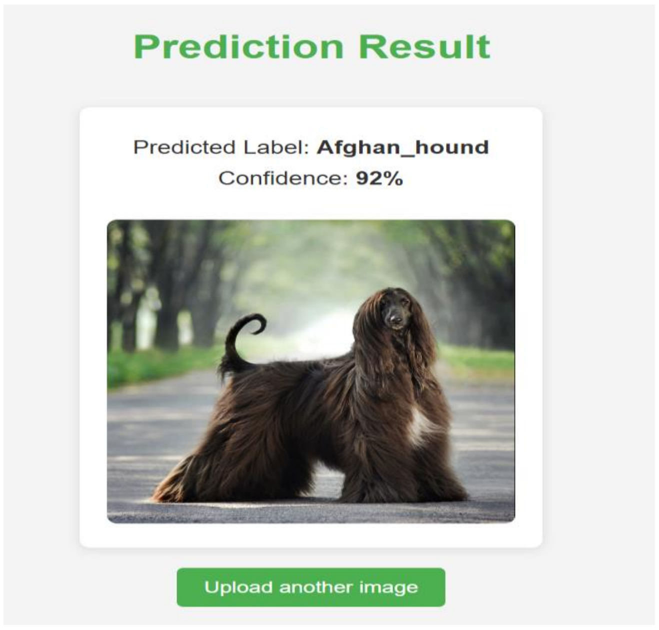


Fig 7.1 screenShot of output

Description:

In this above figure The provided Flask application allows users to upload an image for classification using a pre-trained MobileNetV2 model. When an image is uploaded, it is saved in a designated directory, and the model predicts the image's content. The prediction includes the class label, a human-readable description, and the confidence level of the prediction, rounded to the nearest whole number.


Prediction Result

Predicted Label: **Afghan_hound**
Confidence: **92%**

Fig 7.2 Prediction of Input

Description:

In this above figure The provided Flask application allows users to upload an image for classification using a pre-trained MobileNetV2 model. When an image is uploaded, it is saved

CHAPTER 8
TESTING

The software development life cycle (SDLC) is a conceptual model used in project management that describes the stages involved in an information system development project from an initial feasibility study through maintenance of the completed application. Software development life cycle is a process used by the software industry to design, develop and test the software. Phases in Software Development

- Requirement Analysis
- Software Design
- Development or coding
- Testing
- Maintenance

Testing itself has many phases i.e is called as STLC. It includes

8.1 Test Plan:

It is a document that describes the testing environment, purpose, scope, objectives, test strategy, schedules, milestones, testing tool, roles and responsibilities, risks, training, staffing and who is going to test the application, what type of tests should be performed and how it will tracks and attacks.

Test Development: Preparing test cases, test data, preparing test procedure, preparing test scenario, writing test script.

Test Execution: In this stage, we execute the documents those are prepared I test development phase.

Analyze Result: Once executed documents will get results either pass or fail. We need to analyze the result during the phase.

Defect Tracking: Whenever we get defect on the application, we need to report the bug report file and forwards to test lead team and Dev team. Dev team will fix the bug. Testing an AI hand gesture volume controller involves several steps to ensure that the system correctly interprets hand gestures and adjusts the volume accordingly. Here's a structured approach to testing the functionality:

8.2 Testing Procedure

Define Project Scope and Requirements

Objective: Determine the specific features, such as tracking predicting image , and. User Input:image

Set Up the Development Environment

Tools: Install necessary tools like Node.js, a code editor, and package managers.
Libraries: Install any required libraries (e.g., React for frontend, Express for backend).
Version Control: Set up a Git repository to manage the codebase.

Create the Data Structure

mobileNetV2 pretrained model

Build the Core Logic

Image_prediction using mobileNetV2 pretrained model by flask application.

.

1. **Develop the User Interface (UI)**
   - Input Form: Create a form where users can input or upload img sources
   - Output Area: prediction scores in terms of X and Y
2. **Implement Event Handling**
   - Form Submission: Add event listeners to handle form submissions.
   - Process Input: Pass user input to the core functions and display the results.
3. **Add Error Handling**
   - Input Validation: Ensure that user input is valid (e.g., non-empty, valid characters).
   - Error Messages: Provide feedback for invalid input
4. **Test the Application**
   - Unit Testing: Test individual functions using frameworks like Jest.
   - Integration Testing: Test the interaction between the input form, data processing, and result output.
   - Manual Testing: Run the application in various scenarios to ensure it works as expected.
5. **Optimize and Refactor**
   - Code Review: Review the code for optimization opportunities (e.g., improving performance, readability).
   - Refactoring: Clean up and refactor the code as needed to ensure maintainability.

6. **Deploy the Application**
    - Hosting: Choose a hosting platform (e.g., GitHub Pages, Netlify, or a custom server).
    - Deployment: Deploy the application and make it available to users.
    - Testing in Production: Monitor the deployed application for any issues or bugs**.**
7. **Continuous Improvement**
    - User Feedback: Gather feedback from users and make necessary improvements.
    - Feature Enhancements: Consider adding more features, such as a more extensive nutritional database or personalized exercise recommendations.

8.3 Testing Type

1. **Unit Testing**
    - Purpose: Test individual functions or components of the wellness and nutrition monitoring system.
    - Tools: Use JavaScript testing frameworks like Jest, Mocha, or Jasmine.
    - Steps:
        - Identify key functions (e.g., nutritional value retrieval, BMI calculation, exercise recommendations).
        - Write test cases for each function, covering typical, edge, and invalid cases.
        - Mock dependencies if needed to isolate tests.
        - Run tests and ensure all pass before proceeding.
2. **Integration Testing**
    - Purpose: Ensure that different components work together as expected.
    - Tools: Can be done using Jest, Mocha, or Cypress (for UI-related integration tests).
    - Steps:
        - Identify key interactions between components (e.g., user input, data processing, result output).
        - Create scenarios that test these interactions (e.g., calculating BMI after entering weight and height).
        - Validate that the output and behavior are correct when components work together.
3. **End-to-End Testing**
    - Purpose: Test the entire system as a user would interact with it.
    - Tools: Cypress, Selenium, or Puppeteer for automating browser-based interactions.
    - Steps:
        - Define user stories (e.g., a user enters a food item and receives nutritional values, BMI, and exercise suggestions).
        - Automate the input of these user stories in the web interface.

8.4 Test Case

Testing is evaluating the software to check for the user requirements. Here the software is evaluated with intent of finding effects. It involves executing an implementation of the software with test data and examining the outputs of the software and its operational behaviour to check that it performing as required.

Test case Number	TC_03
Module Under Test	Unit Testing
Description	The error handling for missing files and invalid file types works as expected.
Output	The output of this test case is to determine whether the prediction of image by the user are being correctly obtained
Remarks	Test Successful

The above table represents the Test Case 01. The module represents the unit testing where the user first selects their specific food item. Next, they choose the category of food they intend to get nutritional values. upon clicking the search button, the system processes the input and provides output on food nutritional values successfully.

CHAPTER 9
CONCLUSION AND FUTURE SCOPE

In this project, we developed a web application that leverages the power of deep learning for image classification using the MobileNetV2 model. The application allows users to upload images and receive real-time predictions about the content of those images, showcasing the integration of advanced machine learning techniques with user-friendly web interfaces.Key aspects of the project include:

1. **User Interaction**: The application provides a simple and intuitive interface for users to upload images. By utilizing HTML forms and Flask's routing capabilities, we ensured a seamless user experience.
2. **Image Processing**: Upon receiving an uploaded image, the application preprocesses it to meet the input requirements of the MobileNetV2 model. This step is crucial for ensuring accurate predictions.
3. **Deep Learning Integration**: We successfully integrated the MobileNetV2 model into the Flask application, allowing it to classify images based on a pre-trained dataset (ImageNet). The model's ability to recognize a wide range of objects enhances the application's versatility.
4. **File Handling and Security**: The application includes robust file handling mechanisms, ensuring that uploaded images are securely processed and stored. We implemented measures to validate file types and sizes, safeguarding against potential vulnerabilities.
5. **Dynamic Results Display**: The application dynamically displays prediction results, including the predicted label and confidence score, providing users with immediate feedback on their uploads.

Overall, this project demonstrates the potential of combining web development with machine learning, making sophisticated image classification technology accessible to users without requiring technical expertise. The application serves as a foundation for further enhancements, such as expanding the model to classify additional categories, improving the user interface, or deploying the application to a cloud platform for broader accessibility.As technology continues to evolve, the integration of AI in everyday applications will likely become more prevalent, paving the way for innovative solutions across various fields. This project exemplifies how deep learning can be harnessed to create practical and impactful applications, contributing to the growing landscape of intelligent systems.

Future Scope

1. **Model Fine-Tuning**: To improve accuracy for specific applications, future work could involve fine-tuning the MobileNetV2 model on a custom dataset. This would allow the model to learn features relevant to the specific classification task.
2. **Support for Additional Formats**: Expanding the application to support more image formats (e.g., GIF, BMP) and allowing for higher resolution images could enhance usability and accessibility.

3. **Enhanced Security Measures**: Implementing more robust security practices, such as deeper validation of uploaded files, implementing rate limiting, and using secure storage for uploaded images, can help mitigate risks.
4. **User Interface Improvements**: Enhancing the user interface with modern design principles, responsive layouts, and user-friendly features (e.g., drag-and-drop uploads, previews of uploaded images) can significantly improve user experience.
5. **Real-Time Processing**: Exploring options for real-time image classification using webcam input or video streams could broaden the application's use cases, such as in surveillance or interactive applications.
6. **Integration with Other Models**: Future versions of the application could integrate other deep learning models for different tasks, such as object detection (e.g., YOLO, SSD) or segmentation (e.g., U-Net), allowing users to perform more complex analyses on images.
7. **Deployment and Scalability**: Deploying the application on cloud platforms (e.g., AWS, Google Cloud) with scalable infrastructure can enhance performance and accessibility, allowing more users to access the application simultaneously.
8. **User Feedback Mechanism**: Implementing a feedback system where users can report incorrect predictions or provide suggestions can help improve the model and the overall application over time.

REFERENCES

1. Rawat, W., & Wang, Z. (2017). Deep convolutional neural networks for image classification: A comprehensive review. Neural Computation, 29(9), 2352-2449. https://doi.org/10.1162/neco_a_00990
2. Gu, J., Wang, Z., Kuen, J., Ma, L., Shahroudy, A., Shuai, B., ... & Chen, T. (2018). Recent advances in convolutional neural networks. Pattern Recognition, 77, 354-377. https://doi.org/10.1016/j.patcog.2017.10.013
3. Zhu, X. X., Tuia, D., Mou, L., Xia, G. S., Zhang, L., Xu, F., & Fraundorfer, F. (2017). Deep learning in remote sensing: A comprehensive review and list of resources. IEEE Geoscience and Remote Sensing Magazine, 5(4), 8-36. https://doi.org/10.1109/MGRS.2017.2762307
4. Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). Mobilenetv2: Inverted residuals and linear bottlenecks. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (pp. 4510-4520). https://doi.org/10.1109/CVPR.2018.00474
5. Chollet, F. (2017). Building powerful image classification models using very little data. https://blog.keras.io/building-powerful-image-classification-models-using-very-little-data.html
6. Jain, S., Wankhede, S., Patil, S., & Jain, S. (2019). Plant Disease Detection Using Deep Learning. International Journal of Engineering and Advanced Technology (IJEAT), 8(5), 1515-1518. https://www.ijrat.org/downloads/Journal_IJRAT/Vol-8/issue-5/1515-1518.pdf
7. Brock, A., Donahue, J., & Simonyan, K. (2021). Large Scale GAN Training for High Fidelity Natural Image Synthesis. In International Conference on Learning Representations. https://openreview.net/forum?id=4N8V6T3Yq0

These references provide valuable insights into the techniques and methodologies used in image classification and deep learning, and the links direct you to the respective papers or resources for further reading.