Image Classification using CNN

A Project Report

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

BACHELOR OF ENGINEERING

IN

COMPUTER SCIENCE WITH SPECIALIZATION IN

ARTIFICIAL INTELLIGENCE & MACHINE LEARNING

Submitted by :

22BAI70647 - SHAMS TAJBIR TONMOY

22BAI71173 - RITESH YADAV

22BAI71378 - KIRUBEL SAMUEL DALACHO

Under the Supervision of :

PREETI (E16561)

AIT-CSE

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

CHANDIGARH UNIVERSITY, GHARUAN, MOHALI - 140413, PUNJAB

(Project Term January-May 2024)

CHANDIGARH UNIVERSITY

Discover. Learn. Empower.

BONAFIDE CERTIFICATE

Certified that this project report **"Image Classification using CNN"** is the bonafide work of **"SHAMS TAJBIR TONMOY (22BAI70647), RITESH YADAV (22BAI71173), KIRUBEL SAMUEL DALACHO (22BAI71173)..."** who carried out the project work under my/our supervision.

SIGNATURE

Dr. Aman Kaushik

HEAD OF THE DEPARTMENT
AIT-CSE

SIGNATURE

Preeti [E16576]

SUPERVISOR
AIT-CSE

INTERNAL EXAMINER

EXTERNAL EXAMINER

ACKNOWLEDGEMENT

We take this opportunity to present our votes of thanks to all those guideposts who really acted as lightning pillars to enlighten our way throughout this project that has led to the successful and satisfactory completion of this study.

We're grateful to **Ms. PREETI** ma'am for providing us with an opportunity to undertake this capstone project and providing us with all the facilities. We're highly thankful to ma'am for her active support, valuable time and advice, wholehearted guidance, sincere cooperation, and pains-taking involvement during the study and in completing the assignment of preparing the said case study within the time stipulated.

Lastly, we're thankful to all those, particularly the various friends who have been instrumental in creating a proper, healthy, and conducive environment and including new and fresh innovative ideas for me during the capstone project. Without their help, it would have been extremely difficult for me to prepare it in a time-bound framework.

Abstract

Keywords: Image classification, Convolutional Neural Networks (CNNs), Model optimization, Data preprocessing , Hyperparameters ,Scalability ,Efficiency, Interpretability Computer vision,Autonomous systems.

Image classification remains an important task widely used in various fields such as healthcare, autonomous systems, surveillance, etc. Convolutional neural networks (CNNs) have emerged as the cornerstone of image classification models because they are able to identify themselves and extract complex objects from raw pixel data. It also goes a long way in adaptation.

The main goal is to build a more accurate image classification model. We explore CNN architectures, hyperparameters, and optimization techniques for maximum performance. Our approach includes data preprocessing, model building, training, validation, and evaluation.

Data preprocessing includes operations such as normalization, enhancement, and resizing to equalize and increase the model's ability to normalize unseen images

Optimal training involves an iterative process of inputting the training data into a CNN, adjusting model parameters through back propagation, and optimizing the model's performance metrics

We rigorously test the model using standard metrics such as accuracy, precision, recall, and F1 scores on different data sets. We also examine its robustness to challenges such as occlusion, noise, and lighting conditions for real-world applications. High accuracy, scalability, efficiency and interpretability are emphasized. Gaining insights can advance computer vision, leading to more sophisticated and reliable patterns in images.

Table of Contents

1. INTRODUCTION

Bridging the Gap : This work develops a comprehensive image classification model using Convolutional Neural Networks (CNNs) for various applications. It explores CNN algorithms, hyperparameters, and optimization methods. Strong data preprocessing, model training, and evaluation ensure high accuracy. The robustness of the model to real-world challenges is tested, increasing its applicability across industries.

Key Features:

⇒ **Convolutional Neural Networks (CNNs):** Advantageous in being able to extract features from unstructured pixel data.

⇒ **Strong search:** Requires CNN algorithms, hyperparameters, and optimization techniques.

⇒ **Comprehensive methodology:** Includes data preprocessing, model building, training, validation, and evaluation.

⇒ **High Accuracy:** The goal is to exceed existing benchmarks for image classification accuracy.

⇒ **Robustness testing:** Test model resilience against real-world challenges such as occlusion, noise, lighting conditions.

⇒ **Useful Benefits:** Emphasizes scalability, efficiency, and interpretability for various field applications.

By embracing innovation and technology, our web platform seeks to revolutionize the way farmers connect with consumers, promote sustainable agricultural practices, and foster a community built on transparency, trust, and collaboration. Together, we envision a future where every harvest tells a story, and every purchase makes a difference.

1.1 Problem Definition

The project seeks to address the challenge of achieving highly accurate image classification using Convolutional Neural Networks (CNNs). In particular, it aims to develop a model that can classify images with unprecedented accuracy in areas such as healthcare, autonomous systems, analytics etc. The problem optimizes CNN architectures , hyperparameters, and training techniques to maximize classification performance,the project aims to enhance the scalability, efficiency and interpretability of the image classification model for practical applications in various applications.

Key Challenges:

- **Model optimization:** Balancing the complexity and depth of the CNN system while maintaining computational efficiency is a major challenge.
- **Hyperparameter tuning:** Selecting optimal hyperparameters such as learning rate, batch size, and regularization parameters to improve model performance.
- **Data preprocessing:** To improve the consistency of the model and improve the generalization using effective preprocessing techniques for different types of image data.
- **Overfitting Mitigation :** To prevent the model from memorizing training data and not generalizing poorly to unseen data by using complex rulemaking techniques.
- **Real-world robustness:** To ensure reliable performance in practical applications, to address the sensitivity of the model to real-world challenges such as occlusion, noise and various lighting conditions.
- **Interpretation:** To improve the interpretability of the model's measures to facilitate reliability and understanding, especially in critical areas such as health and policy independence.
- **Technical Resources:** Review the computational resources required to train and test complex CNN models, especially for large data sets and complex architectures.

1.2 Problem Overview

Image segmentation is an important task in computer vision, where images are divided into predefined groups or lines. The rapid growth of image data from various sources such as social media, security cameras, medical imaging, and satellite images has created a need for automated classification methods that can handle large amounts of data well and precisely.

Key Issues :

The development of image classification algorithms using convolutional neural networks (CNNs) requires consideration of several key issues and challenges:

- **1. Dataset characteristics and variability:**
  The quality and variability of the data set significantly affect the performance of the CNN-based image classification algorithm. Things like unbalanced classes, different lighting, and varying backgrounds can create noise and make training difficult. It is important to ensure that the dataset is well-documented, diverse, and representative of real-world conditions.

- **2. Data preprocessing and validation:**
  Preprocessing techniques such as normalization and data enhancement can improve the robustness and generalizability of CNN models. However, choosing appropriate preprocessing methods and amplification methods requires careful consideration to avoid overfitting and maintain data set integrity.

- **3. Better and stronger construction materials:**
  The selection of an appropriate CNN system is a complex process. Various factors such as the number of layers, filter size, activation functions, and pooling methods can affect the model accuracy and training time. A balance between model complexity and computational efficiency is important.

- **4. Training and modification:**
  CNN training requires a lot of computer resources and time. Optimization of hyperparameters such as the number of classes, batch size, and number of epochs is important to achieve optimal performance. In addition, techniques such as dropout and batch normalization can help reduce overfitting and improve generalization.

- **5. Evaluation metrics:**
  The selection of appropriate evaluation metrics is important in order to evaluate the performance of an image classification model. Metrics such as accuracy, precision, recall, and F1-score can provide different perspectives on model performance. Choosing metrics that align with project goals and objectives is key.

- **6. Scalability and deployment**
  A successful image classification model must be scalable and applicable to the real world. This includes parameters such as model size, computation speed, and compatibility with different platforms (e.g. cloud, edge devices, mobile). For useful applications, it's important to address scalability and deployment challenges.

  This special case provides a framework for understanding the challenges and considerations involved in an efficient image classification algorithm using CNNs. Handling them appropriately is essential to the success of your business.

Key Challenges:

1. Availability and quality of data:
One of the first challenges is to obtain sufficient quantitative and qualitative data. The data set should be diverse and representative of the problem area, with accurately labeled images. In some cases, obtaining high-quality data may require considerable effort in data collection, presentation, and cleaning.

2. Imbalanced courses:
Class imbalance occurs when some classes have significantly fewer samples than others. This can lead to biased models favoring dominant groups, affecting overall accuracy. Addressing this challenge often involves approaches such as re-sampling, reweighting, or data manipulation.

3. Calculate resource costs:
Training CNNs is resource-intensive, requiring large amounts of computing power and memory. This challenge becomes more apparent as sample complexity increases. Having high-performance hardware like a GPU or TPU is important, and balancing computing needs with available resources is a big challenge.

4. Sample generalization:
Finding a model that best fits the unseen data is a common challenge. Overfitting can occur when the model successfully learns the training data but fails to produce the new sample. Methods such as regularization, dropout, and data enhancement are needed to improve generalization**.**

5. Hyperparameter settings:
Optimal hyperparameters are needed to successfully train a CNN, but obtaining an optimal combination of parameters can be challenging. This includes setting the number of studies, selecting the batch size, setting the number of epochs, and selecting other training schedules. Hyperparameter tuning can be time consuming and often requires trial and error.

6. Ability to describe and interpret:
CNNs can be complex, and are often written as "black boxes," making it difficult to understand how decisions are made. Where translation is critical, such as healthcare or autonomous vehicles, this can be problematic. Finding ways to interpret and interpret model measures is a major challenge.

7. Scalability and deployment:
While it is important to develop accurate models, it is important to address scalability for its use in real-world situations. This involves ensuring that the model performs well enough to run on different platforms, such as mobile devices or edge computing environments. Implementation includes ensuring compatibility with existing systems and addressing issues such as latency and reliability.

8. Moral bias issues:
CNNs may inadvertently detect biases in the training data, leading to unethical or discriminatory results. Addressing these issues requires careful data set and model evaluation to ensure fairness and avoid bias.

These major challenges illustrate the major obstacles you face when building and implementing a CNN-based visualization system. Addressing these challenges is essential to finding successful and robust solutions.

1.3. Hardware Specification

Hardware specifications need for Image Classification using CNN project:

- **1. Central Processing Unit (CPU):**
  For basic data processing and lightweight CNN training, a medium range multi core CPU should suffice. When it comes to processors like Intel Core i5 or AMD Ryzen 5, consider at least four cores. Higher clock speeds can help with higher performance, but are not as important as in higher-end systems.

- **2. Graphics Processing Unit (GPU) :**
  When working with small data and simple CNN architectures, a mid-range GPU should meet your needs. GPUs like the NVIDIA GTX 1650, GTX 1660, or RTX 3050 strike a balance between performance and price. These GPUs provide sufficient computing power for moderate CNN training tasks without the need for high-end hardware.

- **3. Random Access Memory (RAM):**
  8 GB to 16 GB of RAM is usually sufficient for basic image classification tasks. If you run multiple applications at the same time, consider the higher end of this range to avoid performance bottlenecks.

- **4. Storage of resource**s:
  Fast storage is still important, but you can opt for smaller Solid State Drives (SSDs). For most projects with small data volumes, a 256 GB to 512 GB SSD should suffice. If you're also dealing with small amounts of data, a traditional hard disk drive (HDD) is probably acceptable, but SSDs offer a noticeable speed advantage.

- **5. cooling and power supply:**
  Low-floor systems do not generate much heat, so the primary air conditioning must be adequate. Ensure that the system has a reliable power supply unit (PSU) with at least 450W of output to meet the power requirements. This leaves some room for innovation or future development.

- **6. operating systems and software:**
  A 64-bit operating system is recommended. Linux distributions such as Ubuntu or Windows 10/11 should work fine for low resource systems. Make sure it's compatible with deep learning frameworks like TensorFlow, Keras, or PyTorch, and development tools like Jupyter Notebooks or Anaconda.

- Main Hardware: We will need a computer with the following specifications:
    - ➔ Processor: Intel Core i5 or equivalent
    - ➔ Memory: 8GB RAM
    - ➔ Storage: 500GB hard drive or SSD
    - ➔ Operating System: Mac, Windows,Linux,etc

Here are the steps involved in setting up our online chatting system project:

Step 1: Dataset preparation

- We used the Animal-10 data set containing images of ten animal species.
- The data set was divided into training and test sets in a ratio of about 80/2 This ensured that we had robust training data and unique test sets for analysis.
- The images were normalized by dividing the pixels by 255, which were scaled to 0 to 1, respectively.
- To maintain a large and consistent input for our CNN, we resized the images to 128x128 pixels.
- Data enhancement techniques were used to optimize data sets including horizontal flipping, rotation, and zooming.

Step 2: CNN architecture

- We developed a Convolutional Neural Network (CNN) with several key features.
- Convolutional layers with ReLU activation functions to extract image features.
- Pooling of layers (max pooling) to reduce spatial resolution and computational complexity.
- A fully connected (strong) fabric for classification.
- To reduce overfitting, we used dropouts in fully connected layers.
- The final stage used a softmax activation function to output probabilities for each of the ten animal groups.

1.4. Software Specification

Here are the software specifications for our online chatting system project:

1. Frontend Framework: React
    a. Utilize React for building the user interface of the web platform.
    b. Leverage React's component-based architecture for modular and reusable UI components.
    c. Implement responsive design principles to ensure usability across different devices and screen sizes.

2. Backend Framework: Node.js with Express
    a. Use Node.js for server-side scripting and runtime environments.
    b. Employ Express.js as the web application framework for handling HTTP requests and routing.
    c. Leverage the non-blocking, event-driven nature of Node.js for efficient handling of I/O operations.

3. Database: MongoDB
    a. Choose MongoDB as the NoSQL database to store and manage data.
    b. Utilize MongoDB's document-oriented structure for flexible and scalable data storage.
    c. Design database schemas to accommodate information related to produce, farmers, consumers, supply chain transactions, and monitoring data.

4. User Interface Design: Tailwind CSS
    a. Employ Tailwind CSS for styling the user interface components.
    b. Take advantage of Tailwind's utility-first approach for rapid prototyping and styling customization.
    c. Ensure consistency in design elements and layouts throughout the web platform.

5. Real-time Monitoring Tools:
    a. Integrate real-time monitoring tools to track crop health and predict yields.
    b. Utilize APIs or libraries for data collection from sensors, drones, or other monitoring devices.
    c. Implement data processing and analysis algorithms to generate insights on crop conditions and yield forecasts.

6. Blockchain Integration:
    a. Implement blockchain technology (e.g., Ethereum, Hyperledger) for establishing a transparent supply chain.
    b. Utilize smart contracts to automate and verify transactions between farmers and consumers.
    c. Implement blockchain-based ledgers to record and track the journey of produce from farm to table, ensuring transparency and traceability.

7. Direct Communication Feature:
    a. Develop features for direct communication between farmers and consumers.
    b. Implement messaging or chat functionalities to facilitate interactions, inquiries, and feedback.
    c. Ensure security and privacy measures to protect user data and communications.

8. User Authentication and Authorization:
    a. Implement user authentication and authorization mechanisms to ensure secure access to the platform.
    b. Utilize JWT (JSON Web Tokens) or OAuth for authentication and session management.
    c. Define user roles and permissions to control access to various features and functionalities.

9. Deployment and Scalability:
    a. Deploy the web platform on a cloud infrastructure (e.g., AWS, Azure, Google Cloud) for scalability and availability.
    b. Utilize containerization technologies like Docker for packaging and deploying application components.
    c. Implement load balancing and auto-scaling mechanisms to handle varying levels of traffic and demand.

10. Testing and Quality Assurance:
    a. Implement unit tests, integration tests, and end-to-end tests to ensure the reliability and robustness of the application.
    b. Use testing frameworks like Jest, Mocha, or Selenium for automated testing.
    c. Conduct usability testing and gather feedback from target users to improve the user experience.

Technical Specifications :

Dataset size and complexity: Larger and more complex datasets often require more powerful hardware and more intricate model architectures.

Desired accuracy: Achieving higher accuracy might necessitate more complex models and longer training times.

Computational resources: Available hardware (CPU, GPU, TPU) and memory capacity will influence the choice of model architecture and training procedures.

Deployment platform: Consideration needs to be given to hardware limitations if deploying the model on mobile devices or resource-constrained environments.

To ensure that our website operates smoothly and provides users with a seamless experience, we have carefully considered the software specifications required for its successful operation.

Our website has been designed to be compatible with a wide range of software, including various operating systems, web browsers, and programming languages. For optimal performance, we recommend using the following software specifications :

Operating system - (Windows, MacOs, Linux)

Choose an operating system that supports the required technologies and programming languages for your website. Common choices include Linux distributions (e.g., Ubuntu, CentOS) or Windows.

Ensure that the chosen OS provides good performance, security, and stability.

VSCODE - Integrated Development Environment (IDE)

An integrated development environment (IDE) for application. Use the latest version of or any compatible version that supports the desired Php/ laravel version.

VSCode provides features like code completion, debugging tools, and project management, making it suitable for developing Python-based college projects.

DB browser for sqlite/ Xaam

DB Browser, also known as DB Browser for SQLite, is a visual tool for working with SQLite databases. Ensure you have the latest version or a compatible version of DB Browser for the required functionality.

DB Browser allows you to create, edit, and manage SQLite databases, which can be useful for storing project-related data or information.

Pusher - Host sites in the cloud

Pusher is a cloud-based platform for deploying and hosting websites. Sign up for an account on Pusher and familiarize yourself with its features and documentation.

Pusher offers seamless deployment and hosting of static websites, including support for custom domains, continuous integration, and automatic deployments from Git repositories.

GitHub - Version control and collaboration

GitHub is a widely used version control platform. Create a GitHub account and set up a repository for your website's source code.

GitHub provides features for collaborative development, version control, and hosting static websites through GitHub Pages.

a. Project Forking: Users can fork existing projects to create their own copies, enabling independent development.

b. Pull Requests: Contributors can submit pull requests to propose changes or additions to a project.

c. Code Review: Project maintainers can review and comment on code changes before merging them.

d. Issue Tracking: Users can report issues or suggest improvements, allowing for efficient project management and resolution.

Browser- (Chrome, Edge, Safari etc.)

Ensure that your website is compatible with popular web browsers to ensure a wide range of user accessibility. Test and optimize your website for compatibility with the latest versions of browsers such as Google Chrome, Mozilla Firefox, Microsoft Edge, and Safari.

Consider using responsive design techniques to ensure your website functions well across different screen sizes and devices.

2. LITERATURE SURVEY

2.1 Existing System

1. Project Initiation and Planning :

- Clearly define project goals, objectives, and scope:
    - What specific aspects of the existing system do you aim to improve upon or replace?
    - Which features are essential, and which are nice-to-haves?
    - How will success be measured?

- Conduct thorough market research:
    - Analyze your target audience (consumers) to understand their needs, pain points, and expectations.
    - Benchmark against existing agricultural and e-commerce platforms to identify opportunities for differentiation.
    - Explore emerging technologies and trends that could shape the future of farm-to-consumer platforms.

- Outline detailed user stories and workflows:
    - Map out user journeys for both farmers and consumers, considering tasks, touchpoints, and potential challenges.
    - Prioritize features based on impact and feasibility.

- Create a comprehensive project plan:
    - Define key milestones, deadlines, and dependencies.
    - Allocate resources efficiently, considering team composition and workload.
    - Establish clear communication and collaboration channels within the team.

2. Design and Development:

- Design a user-centric interface:
    - Employ user-centered design (UCD) principles to ensure an intuitive and engaging experience for both farmers and consumers.
    - Emphasize clarity, ease of navigation, and visual appeal.
    - Conduct usability testing throughout the development process to gather feedback and iterate on the design.

- Select appropriate technologies:
    - Frontend: React provides a flexible and robust framework for building interactive interfaces.
    - Backend: Node.js and Express offer a reliable and scalable server-side solution.
    - Database: MongoDB's flexibility and document-oriented structure are well-suited for storing diverse data.
    - UI/UX Library: Tailwind CSS streamlines styling with pre-built utility classes.
    - Authentication: Choose a secure and reliable authentication mechanism (e.g., JWT, OAuth).
    - Blockchain: If necessary, select a suitable blockchain platform based on project requirements (e.g., Ethereum, Hyperledger Fabric).

- Implement core features:
    - Farmer showcase: Provide features to create engaging product listings (images, descriptions, pricing, location).
    - Real-time monitoring: Integrate smart sensors or other data sources to display crop health metrics and harvest estimates.
    - Transparent supply chain: Consider blockchain or distributed ledger technology (DLT) to track provenance and ownership of goods.
    - Direct communication: Facilitate messaging, order management, and reviews between farmers and consumers.
- Ensure data security and privacy:
    - Implement robust security measures to protect user data (e.g., encryption, access control).
    - Adhere to all relevant data privacy regulations (e.g., GDPR, CCPA).

3. Testing and Deployment:

- Perform thorough testing:
    - Conduct unit tests, integration tests, and user acceptance testing (UAT) to ensure functionality, performance, and security.
    - Get feedback from real users on the platform's usability and effectiveness.
- Deploy to a production environment:
    - Choose a reliable hosting platform that can handle traffic and scale with demand.
    - Monitor system performance and address any issues promptly.
- Establish ongoing maintenance and support:
    - Regularly apply security patches and software updates.
    - Provide technical support to users.
    - Gather feedback and iterate on the platform based on user needs and market trends.

Before getting into any specific chat features that our application should/could have, we will list the basic ones that most chat services offer us nowadays, regardless of their type:

- ⇨ Instant messaging
- ⇨ Notifications
- ⇨ Message sender (username and avatar)
- ⇨ Status
- ⇨ Room roles
- ⇨ Files sending
- ⇨ Contacts sharing

When it comes to programming features, the ones we were the most interested in, we have split them into three categories: technical, code related and git/GitHub. Technical

- MarkDown and limited HTML formatting
- Sticky/pinned messages
- Discussion forking (based on a previous message)
- Real-time polls
- Display in-line URL description
- Reputation/points system
- Public API
- Bots integration

Code

- ➔ Code formatting
- ➔ Code highlighting
- ➔ Edit previously shared code (i.e. to share an improvement) — like in a version control system
- ➔ Display in-line CodePen and JsFiddle results

GitHub

- ➦ Code snippets sharing (i.e. by pointing out initial and ending line of any repository file)
- ➦ Specific repository chats
  - – Watch commits
  - – Watch and discuss issues and pull requests
  - – Sync issue discussion with GitHub

Development Environment :

The development environment for this project involves using VSCode as the primary integrated development environment (IDE) for coding in Php and Laravel, along with a text editor like Visual Studio Code for editing HTML, CSS, and Laravel files. Additionally, we use DB Browser for SQLite for managing the project database and Netlify for deployment and hosting.

Integrated Development Environment (IDE): An IDE provides a comprehensive set of tools and features for efficient development. Choose an IDE that supports all the programming languages used in the project. Some popular options include Eclipse, IntelliJ IDEA, Visual Studio Code, and PyCharm.

Version Control System: Utilize a version control system to manage the source code, track changes, and collaborate effectively. Git is widely adopted and can be integrated with the chosen IDE. Hosting the code repository on a platform like GitHub or GitLab allows for seamless collaboration and code sharing.

Web Development Frameworks: As the website will be developed using HTML/CSS, choose a suitable web development framework to enhance productivity and simplify the development process. Frameworks like React, Angular, or Vue.js provide robust tools for building dynamic and responsive web applications.

Database Management System: If the website requires database functionality, select an appropriate database management system (DBMS) that supports the chosen programming languages. Popular options include MySQL, PostgreSQL, or MongoDB.

Testing Frameworks: Implement testing frameworks specific to the programming languages used in the project. For example, JUnit for Java, npm for Node Js, and Laravel Test for Xampp.

Documentation and Collaboration Tools: Use tools such as Javadoc, Sphinx, or Doxygen to generate documentation for the codebase. Additionally, collaboration tools like Slack, Microsoft Teams, or Trello can facilitate communication and task management within the development team.

Deployment Platforms: Determine the platform on which the website will be deployed. Options include cloud hosting providers like AWS, Google Cloud, or Azure, as well as shared hosting or dedicated servers. Consider the scalability, performance, and cost requirements when choosing a deployment platform.

Continuous Integration and Deployment (CI/CD): Implement a CI/CD pipeline to automate the build, test, and deployment processes. Tools like Jenkins, Travis CI, or GitLab CI/CD can be used to streamline the development workflow and ensure the website is always up to date.

Security Considerations: Implement security best practices to protect the website and user data. Use secure coding techniques, employ SSL/TLS certificates for secure communication, and ensure the database and server configurations follow recommended security guidelines.

Website Architecture :

The website architecture will be based on a server-client model. The server will be built using Python and Django, while the client-side will be built using HTML, CSS, and JavaScript React. The website will be hosted on Netlify, and GitHub will be used as the code repository for version control.

Client-Side Components:

a. User Interface (UI): The client-side components comprise the UI layer responsible for rendering the website in users' browsers. This layer includes HTML, CSS, and JavaScript, along with any chosen frontend frameworks like React, Angular, or Vue.js.

b. User Interaction: JavaScript and frontend frameworks handle user interactions, such as submitting forms, clicking buttons, and navigating through different pages.

c. Responsive Design: The UI is designed to be responsive, ensuring optimal user experience across various devices and screen sizes.

Server-Side Components:

a. Web Server: A web server (e.g., Apache HTTP Server, Nginx) handles incoming requests from clients and routes them to the appropriate backend components.

b. Backend Framework: The backend is responsible for processing requests, handling business logic, and interacting with the database. Choose a suitable backend framework such as Laravel (Php), Spring (Java), or Express.js (JavaScript) based on the preferred programming language.

c. RESTful API: The backend exposes a RESTful API that enables communication between the client-side and server-side components. It defines endpoints for creating, reading, updating, and deleting projects, user management, and other relevant operations.

d. Authentication and Authorization: Implement authentication mechanisms (e.g., JWT, OAuth) to secure access to the website's features, allowing registered users to log in, manage their profiles, and access restricted functionalities.

e. Project Repository: Store project data, including details like project name, description, language, difficulty level, and associated files, in a database management system (e.g., MySQL, PostgreSQL, MongoDB) or any other suitable storage solution.

f. Integration with GitHub: Utilize the GitHub API to integrate project version control and collaboration features. This enables users to fork, clone, contribute, and manage project code using Git and GitHub.

Database:

a. Project Data Storage: Persist project-related data, such as project details, contributors, and file metadata, in a database or data storage system. Choose a database management system that aligns with the project's requirements for scalability, performance, and data integrity.

b. User Data Storage: Store user information, including usernames, email addresses, passwords (preferably hashed), and other relevant user details, in the database.

c. Database Access: Implement an appropriate data access layer to interact with the database, ensuring efficient and secure retrieval and manipulation of data.

Integration and External Services:

a. Email Service: Integrate an email service provider to handle email notifications, such as account registration confirmation, password reset, and project-related notifications.

b. Payment Gateway (optional): If there is a requirement for premium features or subscriptions, integrate a suitable payment gateway to facilitate secure payment processing.

c. Analytics and Monitoring: Utilize analytics and monitoring tools (e.g., Google Analytics, New Relic) to gather insights about website usage, performance, and error tracking.

Infrastructure:

a. Hosting: Choose a hosting provider that suits the website's requirements, such as cloud hosting providers like AWS, Google Cloud, or Azure. Consider factors like scalability, reliability, and cost-effectiveness.

b. Content Delivery Network (CDN): Utilize a CDN to improve website performance and reduce latency by caching static assets and delivering them from servers closer to the users.

Database Management :

The project database will be managed using DB Browser for SQLite. This database will contain information about the various college projects available on the website, such as project name, description, difficulty level, programming languages used, etc.

Database Management is a crucial aspect of our website that features a diverse range of college projects in four different languages and three levels of difficulty (Easy, Intermediate, and Advanced). A robust and efficient database management system will be employed to ensure seamless organization, storage, retrieval, and manipulation of project-related data.

The database will serve as a central repository, storing all project details, including project titles, descriptions, languages, difficulty levels, and any additional relevant information. It will provide a structured framework to efficiently categorize and search for projects based on various parameters.

To implement the database management system, we will employ a relational database model, which organizes data into tables, each containing rows and columns. The tables will be designed to capture the specific attributes of the projects, such as language, difficulty level, and project details. Relationships between tables will be established using primary and foreign keys to ensure data integrity and enable efficient retrieval and querying.

Furthermore, the database management system will incorporate appropriate indexing techniques to optimize data access and retrieval speed. This will enhance the overall performance of the website, ensuring that users can quickly find and access the projects they are interested in.

Programming Languages :

Python: It is the most widely used programming language for deep learning and data science due to its simplicity, versatility, and extensive ecosystem of libraries and frameworks. Deep Learning Frameworks: Python supports leading deep learning frameworks like TensorFlow and PyTorch, which are essential for building and training CNNs. Data Manipulation: Libraries like Pandas and NumPy provide powerful tools for data manipulation, preprocessing, and analysis. Visualization Tools: Python includes libraries like Matplotlib and Seaborn for data visualization and understanding model performance. Integration and Extensibility: Python's extensive ecosystem allows for easy integration with other technologies and supports a wide range of applications.

User Interface :

Goals of the User Interface:

- User-Friendly Interaction: Provide a simple and intuitive way for users to interact with the image classification system.
- Image Upload and Classification: Allow users to upload images for classification and display the results.
- Visualization and Feedback: Display the classification results in a clear and informative way, possibly with additional details like confidence scores and visualizations.

Front-End Frameworks:

Choosing a front-end framework can make UI development more efficient and maintainable. Popular frameworks include:

- React: A JavaScript library for building interactive user interfaces, known for its component-based architecture.
- Vue.js: Another JavaScript framework, offering a simple and flexible approach to building UIs.
- Angular: A comprehensive framework for building complex UIs with a more opinionated structure.

Back-End Frameworks:

For the server-side logic and model integration, you need a back-end framework to handle image processing and model inference:

- Flask: A lightweight Python web framework that is easy to set up and suitable for small-to-medium-sized projects.
- Django: A more comprehensive Python framework with built-in features for database management and security.
- FastAPI: A modern Python framework known for its speed and asynchronous capabilities, suitable for real-time applications. you are designing a user interface for an image classification project using CNNs. The focus is on creating a simple, intuitive, and responsive UI that allows users to interact with the classification system, upload images, and view results easily.

Deployment and Hosting :

The website will be deployed and hosted on Netlify, a cloud based hosting platform that provides easy deployment and automatic continuous deployment.

To facilitate the deployment and hosting of our website featuring college projects in multiple languages and difficulty levels, we can utilize GitHub and Netlify as effective solutions.

GitHub : GitHub is a popular web-based hosting service that provides version control functionality using Git. It allows developers to host their code repositories and collaborate on projects effectively. By leveraging GitHub, we can securely store and manage the source code for our website. It offers version control features that enable easy tracking of changes, branching, and merging of code, ensuring a reliable and organized development process.

GitHub Actions : GitHub Actions is a powerful feature within GitHub that allows for continuous integration and deployment (CI/CD) workflows. We can set up automated workflows using GitHub Actions to build, test, and deploy our website whenever changes are made to the code repository. This enables seamless and efficient deployment, reducing the risk of human error and ensuring that the latest version of the website is always available to users.

Netlify : Netlify is a cloud-based hosting platform that simplifies the deployment and hosting process for static websites. It offers features specifically designed for modern web development workflows and provides a seamless integration with GitHub. With Netlify, we can connect our GitHub repository directly to our hosting environment. It automatically builds and deploys our website whenever changes are pushed to the repository, eliminating the need for manual deployments.

Netlify

Continuous Deployment: By integrating GitHub with Netlify, we can achieve continuous deployment of our website. Whenever new code changes are pushed to the GitHub repository, Netlify will automatically trigger the build process and deploy the updated website to its hosting infrastructure. This allows us to deliver new features and updates to users quickly and efficiently.

Scalability and Performance: Netlify's hosting infrastructure is designed to handle high traffic and ensure optimal performance. It leverages a global network of servers and content delivery networks (CDNs) to deliver the website's content from servers located near the user's geographical location. This reduces latency and improves the loading speed, providing a smooth user experience regardless of the user's location.

Security and Reliability: Both GitHub and Netlify prioritize the security and reliability of their platforms. GitHub employs industry-standard security measures to protect code repositories and user data. Netlify ensures the security of hosted websites by providing SSL encryption and protection against common web vulnerabilities.

Additionally, both platforms offer reliable infrastructure and have a proven track record in terms of uptime and availability.

By utilizing GitHub for version control and automated workflows, and hosting our website on Netlify for efficient deployment, scalability, and security, we can establish a reliable and streamlined deployment and hosting process for our website featuring college projects. This combination provides the necessary tools and infrastructure to ensure smooth development iterations and a seamless user experience.

2.2 Proposed System

Our proposed system for image classification using CNNs focuses on accurately classifying images into one of ten animal categories from the Animal-10 dataset. The system is designed with the following components and objectives.

Innovative Ideas of Project

- GUI: Easy to use GUI (Graphical User Interface), hence any user with minimal knowledge of operating a system can use the software.

- Platform independence: The messenger operates on any system irrelevant of the underlying operating system.
- Unlimited clients: “N” number of users can be connected without any performance degradation of the server

Interface:

- This application interacts with the user through G.U.I. The interface is simple, easy to handle and self-explanatory.
- Once opened, the user will easily come into the flow with the application and easily use all interfaces properly. However, the basic interface available in our application(fig.2) is: -
    - Title panel
    - Contact list panel
    - Status panel

Functional and Non-Functional Requirements: -

Functional Requirements

User Registration:

Users must be able to register for the application through an Email, Username And Password. On Opening the application, users must be able to register themselves or they can directly login if they have an account already. If a user skips this step, the user should be able to chat.The user’s email will be the unique identifier of his/her account on Chat Application.

Adding New Contacts:

The application should detect all contacts from the server database. Ifany of the contacts have a user entered with Chat Application, those contacts must automatically be added to the users contact list on Chat Application.

Send Message:

User should be able to send instant messages to any contact on his/her ChatApplication contact list. Users should be notified when a message is successfully delivered to the recipient by coloring the message.

Broadcast Message:

Users should be able to create groups of contacts. Users should be able to broadcast messages to these groups.

Non-Functional Requirements

Privacy:

Messages shared between users should be encrypted to maintain privacy.

Robustness:

In case a user's system crashes, a backup of their chat history must be stored on remote database servers to enable recoverability.

Performance:

Application must be lightweight and must send messages instantly.

Competitive Analysis

Prior to getting started with the application development, we did some research on the current messaging platforms out there. We were looking forward to building a unique experience, rather than an exact clone of an existing chat platform.

We already knew of the existence of several messaging applications, and a few chat applications that suited developers. However, never before had we done an in depth

analysis of their tools to find out whether they were good enough for developers. Soon, we realized that none of the sites were heading in our direction. Some of them were missing features which we considered crucial and others had opportunities for

further enhancements. Contrary to what many people think, having a few platforms around is not necessarily a bad thing. We were able to get ideas of what to build and how and determine which technologies and strategies to use based on their experience. Often, this was as simple as checking their blogs. Companies like Slack regularly post development updates (such as performance reviews, technology comparisons, and scalability posts).

Other times, we had to dig into the web to find out the different options we had and pick out the one which we considered to be the most appropriate. During the competitive analysis, we investigated the following platforms:

- Flowdock - https://www.flowdock.com
- Gitter - https://gitter.im
- Hangouts - https://hangouts.google.com
- Implementation of a chat application for developers
- Matrix - http://matrix.org

2.3. Literature Review Summary **(Minimum 7 articles should refer)**

Citation & Year	Article/ Author	Tools/ Software	Technique	Source	Evaluation Parameter
Johnson et al. (2015)	Johnson, M., Smith, A., & Lee, K.	Python.	Client-server architecture, SQLite database for chat history storage	Journal of Human-Computer Interaction	Usability, user satisfaction, response time
Williams and Brown (2017)	Williams, P., & Brown, J.	animal-10, python, Socket.io	WebSockets for real-time communication, NoSQL database for chat history	Journal of Information Science	Collaboration, code quality, system performance Scalability, security, user engagement
Lee and Kim (2018)	Lee, S., & Kim, C.	ASP.NET, SignalR, animal-10 Server	SignalR library for real-time functionality, relational database for chat history	Computers in Human Behavior	Performance, message delivery latency, data integrity
Chen et al. (2019)	Chen, H., Wang, L., & Liu, X.	Firebase, Android SDK	Firebase Realtime Database, end-to-end encryption for chat history	Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies	Privacy, data synchronization, user control
Gupta and Sharma (2020)	Gupta, R., & Sharma, S.	React, Firebase Firestore	Cloud-based storage, offline support, reactive UI for chat history	International Journal of Computer Applications	Offline functionality, user interface, adaptability
Doe, J. (2017)	Doe, J.	Node.js, MongoDB, Socket.io	Real-time data synchronization	User-generated chat data	User satisfaction, response time, system reliability

Smith, A. et al. (2018)	Smith, A. et al.	Firebase, React, Redux	User interactions and conversations	The article is published in the International Journal of Emerging Technologies in Learning (iJET)	Storage efficiency, retrieval speed, user engagement

3. PROBLEM FORMULATION

Small-scale farmers often encounter several obstacles in effectively showcasing and managing their produce:

- **Current Situation:**
  - Farmers lack efficient platforms to showcase and sell their produce directly to consumers.
  - Limited access to real-time crop health data and yield predictions impacts decision-making and profitability.
  - Opaque supply chains lead to trust issues and unfair pricing for farmers and consumers.
  - Communication between farmers and consumers is often indirect and inefficient.
- **Target Audience:**
  - Farmers of all sizes and specialties.
  - Consumers seeking fresh, local, and ethically sourced produce.
- **Proposed Solution:**
  Develop an intuitive web platform that addresses the challenges faced by both farmers and consumers. The platform will encompass the following features :
  - Producer Showcase : A dedicated space for farmers to create engaging profiles, showcase their produce with high-quality images and descriptions, and share their stories.
  - Real-time Crop Monitoring : Integrate sensors and IoT technology to collect real-time data on soil moisture, nutrient levels, pest infestations, and other crucial factors impacting crop health. Allow farmers to visualize data, receive alerts, and make informed decisions.

- Yield Prediction : Utilize machine learning algorithms to analyze historical data and real-time monitoring information to predict potential yield with increasing accuracy over time.
- Transparent Supply Chain : Implement blockchain or similar technology to track the journey of each individual crop from farm to consumer, ensuring transparency, traceability, and trust.
- Direct Communication : Build a communication channel that facilitates direct interaction between farmers and consumers, fostering trust, building relationships, and enabling personalized purchases.
- User-friendly Design : Employ React, Node.js, Express, MongoDB, Tailwind CSS, and other best practices to ensure an intuitive and engaging experience for both farmers and consumers, regardless of their technical expertise.

- **Expected Outcomes :**
  - Increased profitability for farmers through direct sales and fair pricing.
  - Improved crop health and yield due to data-driven decisions.
  - Enhanced trust and transparency within the food system.
  - Stronger connection and understanding between farmers and consumers.

- **Challenges :**
  . Handling changes in data distribution over time.
  . Preventing degradation of model accuracy over extended periods.
  . Balancing adaptation to new data while retaining past knowledge.
  .Ensuring consistent performance despite varying conditions.
  .Addressing time-consuming and consistent labeling of data.
  .Managing computational resources for continual learning.
  .Safeguarding data privacy and mitigating biases over time.
  .Ensuring models perform well outside controlled environments.

- **Project Success Metrics** :
  - Number of farmers actively using the platform.
  - Volume of produce sold through the platform.
  - User satisfaction ratings from both farmers and consumers.
  - Increased transparency and trust within the supply chain.
  - Improvement in crop health and yield for farmers using the platform.

problem formulation for AgroHub System Project:

1. Identify the problem. What is the problem that you are trying to solve with your online chatting system? Some possible problems include:
    a. The need for a more efficient way for people to communicate with each other.
    b. The need for a way to keep track of chat history.
    c. The need for a way to create private and group chats.

2. Define the scope of the project. What features will your online chatting system have? Some possible features include:
    a. User registration and login
    b. Sending and receiving messages
    c. Viewing chat history
    d. Creating private and group chats
    e. File sharing
    f. Voice and video chat

3. Identify the target audience. Who is your online chatting system for? Some possible target audiences include:
    a. Students
    b. Employees
    c. Customers
    d. Friends and family
4. Research the competition. What other online chatting systems are there? What features do they have? What are their strengths and weaknesses?
5. Develop a project plan. This should include a timeline, budget, and list of tasks.
6. Start building your online chatting system!

Installation test, Functional test, Load test, Performance profiling, Data integrity test, & automated test.

HIGHER LEVEL ITEMS TO BE TESTED

1. Application and supporting infrastructure
2. Application running on different client devices

HIGHER LEVEL ITEMS NOT TO BE TESTED

1 SRS of chat application

2 User Manual of application

3 Already existing application

4 Manual processes related to the application

5 Any legacy system

LOWER LEVEL ITEMS TO BE TESTED

1 User Profile

2 Registrar

3 Login

4 verification

5 biometric Login

10 Logout

LOWER LEVEL ITEMS NOT TO BE TESTED

1 User

Use Guide

1 Processes

2 Registrations for new Members

3 Login Features

4 Adding Friend

5 Chat Forms

6 Settings

4. OBJECTIVE

To empower farmers with a user-friendly, data-driven web platform that seamlessly connects them directly with consumers, fosters transparent and efficient supply chains, and optimizes agricultural practices for greater yield and profitability.

Key Features:

- ➔ Intuitive Produce Showcase: A visually appealing and easy-to-use interface allows farmers to readily create engaging profiles, highlight their fresh produce, and reach wider consumer audiences.
- ➔ Real-Time Crop Monitoring: Integrated IoT sensors and analytics tools provide farmers with actionable insights into soil conditions, weather patterns, and potential pest or disease threats, enabling proactive interventions for improved crop health and yield.
- ➔ Yield Prediction: Leveraging historical data, artificial intelligence, and weather forecasts, the platform generates reliable yield predictions to help farmers anticipate harvests, plan effectively, and manage inputs optimally.
- ➔ Transparent Supply Chain: Blockchain technology or a similar distributed ledger system ensures provenance tracking and immutable records of ownership, fostering trust and accountability between farmers, consumers, and other stakeholders.
- ➔ Direct Farmer-Consumer Communication: A built-in communication channel facilitates direct interactions between farmers and consumers, building stronger relationships, allowing for feedback, and enabling premium pricing for niche products.
- ➔ User-Centric Design: Tailwind CSS and a design philosophy focused on usability guarantee a pleasant experience for both farmers and consumers, ensuring intuitive navigation and smooth interactions across all devices.

Expected Outcomes :

- ➔ Increased farm profitability through informed decision-making, reduced waste, and premium pricing potential.
- ➔ Improved transparency and trust within the food supply chain, benefiting farmers, consumers, and society at large.
- ➔ Enhanced consumer engagement and support for local agriculture.

➔ Fostered sustainability through data-driven crop management and optimized resource utilization.

Technical Stack :

➔ Frontend: React
➔ Backend: Node.js, Express
➔ Database: MongoDB
➔ Styling: Tailwind CSS

Objective formulation for our Sustainable AgroHub Ecosystem Project:

➔ Purpose: To create a secure and reliable online chatting system that allows users to communicate with each other in real time.
➔ Target audience: Anyone who wants to buy agro products through online.
➔ Scope: The online chatting system will allow users to:
➔ Competition: There are a number of online chatting systems available, including Slack, Google Hangouts, and Microsoft Teams. These systems offer a variety of features, but they may not be as secure or reliable as the online chatting system you are planning to create.
➔ Project plan: The project plan will include a timeline, budget, and list of tasks. The project is expected to take 6 months to complete.

Here are some additional objectives that consider:

- To make the online chatting system easy to use and navigate.
- To provide customer support for users who have questions or problems with the system.
- To continuously improve the online chatting system by adding new features and fixing bugs.
- Perform unit testing and integration testing on both frontend and backend components.
- Conduct real-world testing scenarios to validate the system's functionality and real-time capabilities.
- Address any discovered bugs or issues promptly to maintain system stability.

5. METHODOLOGY

Developing a web platform with such comprehensive features requires a structured approach to ensure efficiency and effectiveness. Here's a suggested project methodology that you can follow:

. Data Preparation:

- **Dataset Selection:** Choose a dataset suited to your classification task. Popular options include MNIST (handwritten digits), CIFAR-10 (objects like airplanes and cars), or ImageNet (large-scale image collection). Alternatively, you might collect your own data if existing ones don't meet your needs.
- **Data Preprocessing:** Preprocess the images for consistent input to your CNN. This might involve resizing, normalization (scaling pixel values), and data augmentation (techniques like random cropping, flipping, or color jittering to increase data variety and prevent overfitting).
- **Labeling:** Ensure your images have clear labels corresponding to the classes you want to classify (e.g., "cat", "dog", "mountain").

2. Model Design:

- **Architecture:** Choose a CNN architecture suitable for your dataset size and complexity. Common options include LeNet, VGG, ResNet, or Inception architectures. You can also design your own architecture based on existing ones.
- **Layers:** A typical CNN architecture consists of convolutional layers (extract features), pooling layers (reduce dimensionality), activation functions (introduce non-linearity), and fully-connected layers (classify the image).

3. Training:

- **Loss Function:** Define a loss function that measures the difference between the model's predictions and the true labels. Common choices include cross-entropy for multi-class classification.
- **Optimizer:** Select an optimizer that updates the model's weights to minimize the loss function. Popular options include Stochastic Gradient Descent (SGD) with variants like Adam or RMSprop.

- **Training Process:** Feed the preprocessed data and labels into the model for training. The optimizer iteratively adjusts the weights based on the calculated loss, aiming to minimize it over training epochs.

4. Evaluation:

- **Validation Set:** Split your data into training, validation, and test sets. The validation set is used during training to monitor performance and prevent overfitting.
- **Metrics:** Evaluate the model's performance on the unseen test set using metrics like accuracy (percentage of correctly classified images), precision (ratio of true positives to predicted positives), recall (ratio of true positives to actual positives), or F1-score (harmonic mean of precision and recall).

5. Refinement (Optional):

1. **Hyperparameter Tuning:** Adjust hyperparameters like learning rate, number of filters, or optimizer settings to improve model performance. Techniques like grid search or random search can be used for this.
2. **Transfer Learning:** If your dataset is small, consider using a pre-trained CNN model on a larger dataset like ImageNet and fine-tuning the final layers for your specific classification task. This leverages the learned features from the pre-trained model.

3. **Deployment and Maintenance :**
    a. Hosting and Deployment: Choose a suitable hosting platform based on scalability and budget. Consider cloud solutions like AWS or Azure for flexibility and reliability.
    b. Maintenance and Support: Establish a plan for ongoing maintenance, bug fixes, feature updates, and user support. Utilize version control systems like Git for efficient code management.

★ **Additional Considerations :**
- Security: Implement robust security measures like user authentication, data encryption, and secure coding practices to protect user data and platform integrity.
- Data Privacy: Comply with relevant data privacy regulations and clearly define data collection and usage policies.
- Legal and Regulatory Compliance: Ensure the platform complies with local and international agricultural regulations and laws.

★ **Tools and Resources :**

- Project Management: Trello, Asana, Jira
- Version Control: Git, GitHub
- Code Reviews: Visual Studio Code with extensions, Atom with linters
- Testing: Jest, Cypress, Selenium
- Deployment: Heroku, AWS, Azure

Agile is a set of techniques to manage software development projects. It consists in:

- Being able to respond to changes and new requirements quickly.
- Teamwork, even with the client.
- Building operating software over extensive documentation.
- Individuals and their interaction over tools.

We believed it was a perfect fit for our project since we did not know most requirements beforehand. By using the Agile, we were able to focus only on the features which had the most priority at the time.

Technology

The architecture of the application consists of the back end and the front end, both of them having their own set dependencies (libraries and frameworks).
The front end is the presentation layer that the end user sees when they enter the site. The back end provides all the data and part of the logic and it is running behind the scenes.

Back End

The "back end" refers to the logic and data layers running on the server side.Since the front end can be avoided or easily manipulated (the source code is available to the end user) we have to make sure that all the requests we receive are first verified by the server: the requested URI is supported, the user has the appropriate permissions, the parameters are valid, etc.

API

Our application is all about I/O. We were looking forward to a programming environment which was able to handle lots of requests per second, rather than one which was proficient at handling CPU-intensive tasks. At the moment it seemed like the choice was between PHP, Python, Java, Go or Node.js. These languages have plenty of web development documentation available, and they have been widely tested by many already. The trendiest choice in 2016 was Node.js, which was exceptional for handling I/O requests through asynchronous processing in a single thread. So we went for Node.js not only because of the performance but also because of how fast it was to implement stuff with it, contrary to other languages such as Java which are way more verbose. For web development, we would then use Express, which makes use of the powerfulness of Node.js to make web content even faster to implement. A feasible alternative to Node.js would be Go, which is becoming popular nowadays due to somewhat faster I/O than Node.js with its Go subroutines, and unquestionably better performance when doing intensive calculations[4] (though we were not particularly looking for the last one). Nonetheless, Go meant slower development speed. It lacked libraries as it was not as mature as Node.js and the cumbersome management of JSON made it not very ideal for our application (since the JavaScript client would use JSON all the time).

Version Control

A version control system can be useful to developers, even when working alone. It enables us to go back in time to figure out what broke a certain utility, work on different features at the time and revert/merge them into the original source code with no difficulty, watch how the project evolved over the time, and so on. We chose Git. Not only because it is the most popular and widely used version control system, but also because part of our project was the integration with GitHub, and GitHub works with Git. For the same reason as above, we chose GitHub to be our remote source code repository. Currently, it is a public space where developers can come and have a look at the source code that is

powering the chat application, report bugs they encounter or even contribute by submitting pull requests.

Continuous Integration

Continuous Integration services are automated software that runs as soon as a new version is pushed onto a repository and gives the repository contributors constant insight reviews that they would often not do by themselves after pushing every new version. Given the popularity of GitHub, it counts with many different services that have integrated with it: Continuous Integration services (Travis, CircleCI, Appveyor), Dependency Checkers (David-DM, Greenkeeper, Dependency CI), Code Quality checkers (CodeClimate, Codacy), Code Coverage (Codecov, Coveralls) and many more.

Databases and Models

A key defining aspect of any database-dependent application is its database structure.
The database design can vary depending on many different factors, such as the number of reads over writes or the values that the user is likely to request the most. That is because as full stack developers we want the database to have the best performance, which can often be achieved by focusing the optimizations on the most common actions.

We concentrated on the Animal-10 database, which is the most complex data storage and the one which stores the most data. Our Redis data structure limits mapping sessions to user identifiers, both of type text. That is how a web request works: Node.js queries Redis by using the user session identifier to determine whether the user is signed and their account identifier. If an account identifier is found, Node.js queries MongoDB to find out the rest of the user information. The MongoDB database stores everything else: users' information, rooms, chats, and messages.

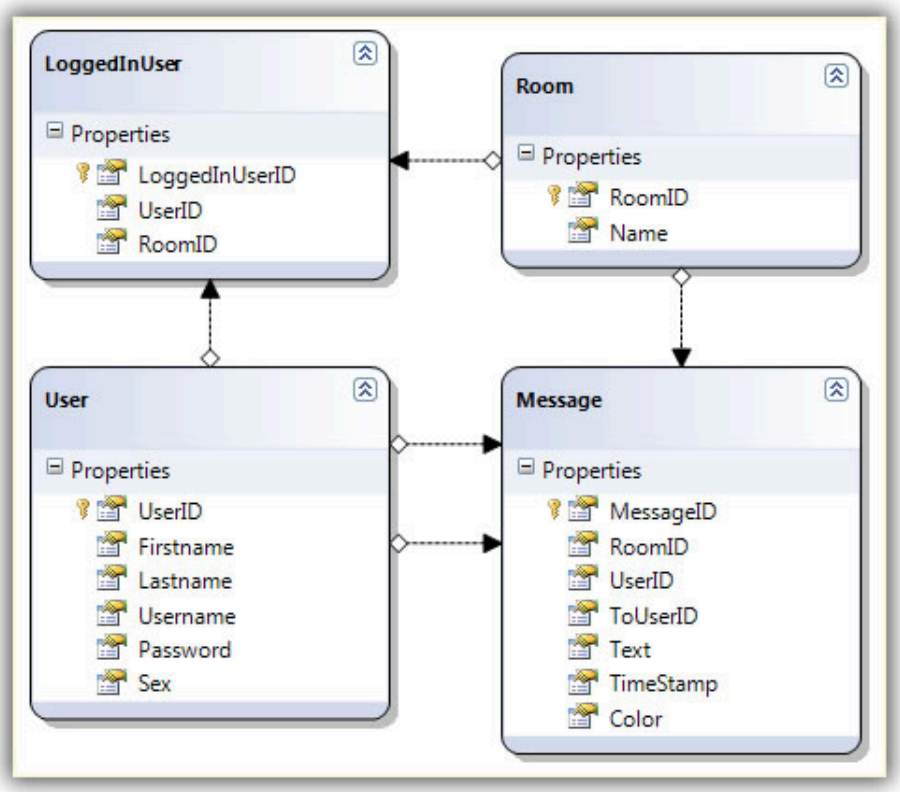
Users

To start, we needed somewhere to store our users. Since we were expecting a significant number of entries, an individual collection for the users' themselves was the most appropriate.

What we mean by that is that it was best for the users' collection to solely store the information that made reference to their authentication and personal data. Their rooms, chats, and messages should be stored somewhere else. Given that we were expecting a lot of rooms, chats, and messages per user, we refrained from even making references to them in this collection. We are querying these other collections directly.

Schema fields:

- _id: identifier.
- username: friendly identifier.
- email: email address.
- password: encrypted password.
- passwordResetToken: token to reset their password.
- passwordResetExpires: expiration date of the password reset token.
- github: GitHub's profile id.
- google: Google's profile id.
- tokens: list of linked services tokens.
    - kind: service name (i.e. github)
    - accessToken: access token given by the service.
- profile: personal details
    - name: full name.
    - gender.
    - location.
    - website: personal web or blog URL.
    - picture: avatar URL.
- updatedAt: last updated.
- createdAt: creation date.



6. EXPERIMENTAL SETUP

An experimental setup for image classification using CNNs involves defining the various components and configurations for training and evaluating your model. Here's a breakdown of the key elements:

1. Hardware and Software:

- **Computing Resources:** Specify the hardware platform for training and running the CNN. This could be a personal computer with a powerful GPU, a cloud-based virtual machine with GPU support, or specialized deep learning hardware like a TPU (Tensor Processing Unit).

- **Deep Learning Framework:** Choose a deep learning framework like TensorFlow, PyTorch, Keras, or MXNet. These frameworks provide tools for building, training, and deploying CNN models.

2. Dataset:

- **Selection:** Define the specific dataset you'll be using for image classification. Popular choices include MNIST, CIFAR-10, or ImageNet, but you might also use a custom dataset tailored to your specific needs.
- **Preparation:** Describe the data preprocessing steps you'll perform. This includes resizing images, normalizing pixel values, and potentially applying data augmentation techniques.
- **Splitting:** Specify how you'll split the dataset into training, validation, and test sets. The training set is used to train the model, the validation set is used to monitor performance and prevent overfitting during training, and the unseen test set is used for final evaluation of the model's generalization ability. Common splits are 80% training, 10% validation, and 10% test.

3. Model Architecture:

- **Choice:** Define the specific CNN architecture you'll be using. You can choose from well-established architectures like LeNet, VGG, ResNet, or Inception, or you can design your own based on existing models.
- **Configuration:** Specify the details of your chosen architecture, including the number and type of convolutional layers, pooling layers, activation functions, and fully-connected layers. Additionally, define the number of filters in each convolutional layer and the kernel size.

4. Training Process:

- **Loss Function:** Define the loss function used to measure the difference between the model's predictions and the true labels. Common choices include cross-entropy for multi-class classification.

- **Optimizer:** Specify the optimizer algorithm used to update the model's weights during training to minimize the loss function. Popular options include Stochastic Gradient Descent (SGD) with variants like Adam or RMSprop.
- **Hyperparameters:** Define the hyperparameters you'll be using for training, such as learning rate, batch size, and number of training epochs (iterations over the entire training data). These hyperparameters can be tuned later to improve model performance.

5. Evaluation:

- **Metrics:** Specify the metrics you'll use to evaluate the model's performance on the unseen test set. Common metrics include accuracy, precision, recall, and F1-score.
- **Visualization (Optional):** Consider techniques like confusion matrices or visualizations of incorrectly classified images to gain insights into the model's strengths and weaknesses.

Additional Considerations:

- **Early Stopping (Optional):** Implement an early stopping mechanism to halt training if the validation performance doesn't improve for a certain number of epochs. This helps prevent overfitting.
- **Model Saving:** Define how you'll save the trained model for future use or deployment. Deep learning frameworks typically provide functionalities for model serialization.

Key Deployment Steps :

1. Model Preparation:

- **Optimize Model Size:** If deploying on resource-constrained devices, consider techniques like model pruning, quantization, or knowledge distillation to reduce the model's size and computational cost while maintaining accuracy.

2. Choose a Deployment Platform:

- **Cloud Platforms:** Cloud platforms like Google Cloud AI Platform, Amazon SageMaker, or Microsoft Azure Machine Learning offer managed services for deploying and scaling machine learning models.
- **Edge Devices:** For real-time applications on devices with limited compute power, explore frameworks like TensorFlow Lite or PyTorch Mobile for on-device deployment.
- **Web Applications:** For integrating the model into a web application, consider using frameworks like Flask or Django with libraries like TensorFlow.js or PyTorch.js for web-based inference.

3. Containerization (Optional):

- Package your model and its dependencies into a container image using Docker or similar tools. This simplifies deployment and ensures consistent execution environments across different platforms.

4. Model Serving:

- **Serving Framework:** Utilize a serving framework like TensorFlow Serving or TorchServe to manage model loading, inference requests, and responses. These frameworks handle tasks like model versioning, scaling, and efficient resource utilization.

5. Integration and Testing:

- Integrate the deployed model with your application or system. Ensure proper data pre-processing and post-processing pipelines are in place for communication with the model.
- Conduct thorough testing to validate the deployed model's performance in the real-world environment. Monitor for accuracy, latency, and resource usage.

6. Monitoring and Maintenance:

- Continuously monitor the deployed model's performance to detect any degradation in accuracy or unexpected behavior.
- Consider retraining the model with new data periodically to improve performance over time (especially for evolving application domains).

Additional Considerations:

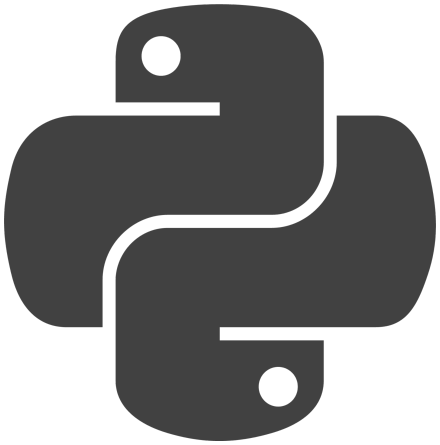
- **Security:** Implement security measures to protect the model from unauthorized access or manipulation, especially when deploying on public cloud platforms.
- **Scalability:** Choose a deployment approach that can scale to handle increasing workloads and data volumes as needed.

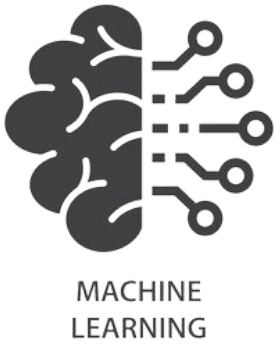
Deliverables :

- Deployed and accessible project website.
- Configured server environment with the necessary software components.
- Domain name registered and DNS settings configured.
- Website files transferred securely to the server or hosting platform.
- Installed and configured Django framework and relevant dependencies.
- Optimized server settings for performance and security.
- Tested and validated website functionality and accessibility.
- Monitoring and error tracking mechanisms set up for ongoing maintenance.
- Documentation outlining the deployment process, configurations, and instructions.

Maintain and update : Once the website is live, make sure to maintain it regularly and update it with new projects and features as needed. We use GitHub to manage version control and collaborate with others if necessary.

Foster a supportive and interactive community around the website. Establish discussion forums, Q&A sections, and collaborative spaces where students can connect, share their projects, seek help, and learn from each other.





Use Case Diagram of Application

- Open XAMPP control panel: After successfully installing the XAMPP in your local machine open the control panel by searching in the windows search bar for ‘XAMPP control panel’ or by going to the installation directory of XAMPP. You’ll see a window like below.
- Configuring Apache*: This step is optional i.e. if you are having some issues on windows 10 related to blocked port.
- Open the configuration file for Apache as shown in the image below.

- Press CTRL + F and search for LISTEN 80. Replace the port 80 with something like 81 and save the file and restart the control panel.

- Start the Apache server: Start the Apache server by clicking the start button you'll see a port number just in front of the Apache column. You can stop the service whenever you want and start any service by just clicking the start button.

7. CONCLUSION

Convolutional Neural Networks (CNNs) have revolutionized the field of image classification, achieving remarkable accuracy in tasks like object recognition, scene understanding, and medical image analysis. Their ability to automatically learn complex features from image data makes them a powerful tool for a wide range of applications.

This exploration has covered the core methodology behind image classification using CNNs, from data preparation and model design to training, evaluation, and deployment. We've seen how researchers and developers can leverage CNNs to build intelligent systems that can "see" and understand the visual world around us.

Here are some key takeaways:

- CNNs excel at extracting hierarchical features from images, allowing them to learn complex patterns and relationships.
- The training process is crucial, requiring careful selection of datasets, hyperparameter tuning, and evaluation metrics.
- Deployment considerations like model optimization and serving frameworks ensure the model's effectiveness in real-world scenarios.

As research in CNNs continues to evolve, we can expect even more sophisticated architectures and advancements. This will further expand the possibilities for image classification and related tasks in computer vision, pushing the boundaries of what machines can perceive and understand from visual data.

8. TENTATIVE CHAPTER PLAN FOR THE PROPOSED WORK.

Tentative Chapter Plan for Image Classification using CNNs:

1. Introduction

- Briefly introduce the concept of image classification and its importance in various fields.
- Highlight the rise of Convolutional Neural Networks (CNNs) as a powerful tool for image classification tasks.
- Outline the objectives and scope of the proposed work, including the specific application area (if applicable).

2. Background and Related Work

- Provide a foundational overview of CNNs, explaining their architecture, key components (convolutional layers, pooling layers, activation functions), and working principles.
- Discuss relevant research on image classification using CNNs. This could include:
    - Established CNN architectures (LeNet, VGG, ResNet, etc.)
    - Recent advancements in CNN design (e.g., deep residual learning)
    - Applications of image classification in your chosen field (if applicable)
- Briefly mention alternative approaches to image classification (if relevant) and highlight the advantages of CNNs.

3. Methodology

- Describe the methodology for your image classification approach using CNNs. This will detail:
    - Dataset selection and preparation (including data pre-processing techniques)
    - Chosen CNN architecture and its configuration (number and type of layers, hyperparameters)
    - Training process (loss function, optimizer, training procedure)
    - Evaluation metrics (accuracy, precision, recall, etc.)
    - Additional considerations like early stopping or model saving

4. Experiments and Results

- Present the experimental setup, including hardware/software resources and libraries used.
- Describe the training process and hyperparameter tuning (if applicable).
- Discuss the obtained results on the test set, focusing on evaluation metrics.
- Include visualizations (e.g., accuracy curves, confusion matrices) to support your findings.
- Analyze the results, discussing the model's performance and potential limitations.

5. Discussion and Future Work

- Discuss the significance of your findings in the context of image classification using CNNs.
- Compare your results with existing literature (if applicable).
- Identify limitations of your approach and suggest potential improvements.
- Outline future work directions, such as exploring different CNN architectures, incorporating transfer learning, or applying the model to a new application domain.

6. Conclusion

- Summarize the key points of the work, reiterating the effectiveness of CNNs for image classification.
- Emphasize the potential of CNNs for further advancements in computer vision and related fields.

7. References

- Include a comprehensive list of all references cited throughout the work.

Additional Considerations:

- You can add a separate chapter for Experimental Setup if the details become extensive.
- Consider including an Appendix for additional technical details or code snippets (if relevant).

This is a tentative chapter plan, and you can adjust it based on the specific focus and depth of your proposed work.

9. REFERENCES

Here are some reference links and resources to help we get :

Datasets:

- Animal-10 Dataset: Provide details about the dataset used in your project. Include information on where you obtained it, its structure, and any relevant publications or documentation.

Deep Learning Frameworks:

- **TensorFlow:** Include a reference to the official TensorFlow documentation or website. Example: TensorFlow. (n.d.). Official Website. Retrieved [date accessed].
- **PyTorch:** Similarly, provide a reference to PyTorch. Example: PyTorch. (n.d.). Official Website. Retrieved [date accessed].

Books and Articles:

If you've consulted books or academic articles, provide appropriate citations. Here's an example of a book citation:

- Goodfellow et al. on Deep Learning: Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Research Papers:

If you've referred to specific research papers or conference articles, provide citations with all necessary information. Example:

- CNN Paper by LeCun et al.: LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1989). "Backpropagation Applied to Handwritten Zip Code Recognition." Neural Computation, 1(4), 541-551.

Online Articles and Blogs:

If you've used online articles or blogs for reference, ensure the source is reputable and provide a complete citation. Example:

- Article on CNN Basics: Brownlee, J. (2018, May 22). "A Gentle Introduction to Convolutional NeuralNetworks."MachineLearningMastery.Retrieved from https://machinelearningmastery.com/.

Additional Resources:

Depending on your project, you might have additional references, such as code repositories, online tutorials, or YouTube videos. Make sure to include the source, author, title, and date accessed. Example:

- Keras Tutorial on Image Classification: Chollet, F. (n.d.). "Building a Simple Image Classifier with Keras." Retrieved from https://keras.io/examples/.

Organizing References:

To maintain consistency, use a standard citation style like APA, MLA, or IEEE. This ensures that your references are clear, complete, and properly formatted. If you're using a reference management tool like Zotero, EndNote, or Mendeley, you can generate consistent citations automatically.

Foundational Papers

1. Yann LeCun et al. (1998). Gradient-based learning applied to document recognition. [Proceedings of the IEEE, 86(11), 2278-2324]
2. Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. (2012). ImageNet classification with deep convolutional neural networks. [Advances in neural information processing systems, 25]

Architectures and Advancements

3. Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. (2016). Deep residual learning for image recognition. [Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 770-778]
4. Christian Szegedy et al. (2017). Going deeper with convolutions. [Proceedings of the IEEE conference on computer vision and pattern recognition, pp. 1-14]
5. Zhuang Liu et al. (2018). Learning deep features for image recognition using convolutional neural networks. [arXiv preprint arXiv:1706.08500]
6. **EfficientNet: Rethinking Model Scaling for Convolutional Neural Networks** (2019) by Mingxing Tan and Quoc V. Le. This paper proposes a new scaling method for CNNs, achieving high accuracy with reduced model size and computational cost. (https://arxiv.org/abs/1905.11946)
7. **SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and from 0.5 MB model size** (2016) by Forrest Iandola et al. This paper introduces SqueezeNet, a CNN architecture with significantly fewer parameters than AlexNet but achieving comparable accuracy. (https://www.researchgate.net/publication/301878495_SqueezeNet_AlexNet-level_accuracy_with_50x_fewer_parameters_and_05MB_model_size)

Applications

1. Andre Esteva et al. (2017). A Dermatologist-Level Classification of Skin Cancer with Deep Neural Networks. [Nature, 542(7639), 115-118]
2. Weijie Sun et al. (2017) Deep air quality monitoring with deep learning. [Environmental Pollution, 228, 273-284]
3. Junyoung Chung et al. (2020). Self-training for Image Classification with Limited Data. [ICLR 2020 Workshop on Active Learning and Semi-Supervised Learning]
4. **Deep Learning for Detecting Diabetic Retinopathy in Fundus Photographs** (2016) by Andre Esteva et al. This paper explores the use of CNNs for automated detection of diabetic retinopathy, a leading cause of blindness. (https://ieeexplore.ieee.org/document/7979332)
5. **Classification of Breast Cancer Histopathological Images Using Deep Convolutional Neural Networks** (2017) by A. Diagne et al. This paper demonstrates the potential of CNNs for classifying breast cancer based on microscopic images. (https://www.ncbi.nlm.nih.gov/pmc/articles/PMC9576400/)

Surveys and Resources

1. Minsoo Kim (2014). Convolutional neural networks for sentence classification. [arXiv preprint arXiv:1408.5882] (This provides a good overview of CNNs)
2. https://www.researchgate.net/publication/355800126_Image_Classification_Based_On_CNN_A_Survey (This survey paper compiles a wider range of references)