CHAPTER – 1

INTRODUCTION

1.1 DOMAIN

The domain of this project is child online safety. In today's digital age, children have unprecedented access to mobile devices, computers, and the internet. While this access offers numerous educational and entertainment opportunities, it also exposes children to potential risks, including exposure to inappropriate or harmful content. The domain of child online safety encompasses various strategies and technologies designed to protect children from these risks. This project focuses on developing a technological solution that uses machine learning and computer vision to monitor and analyze screen content, ensuring a safe online environment for children.

1.2 PROJECT NEEDS

The primary need for this project arises from the increasing exposure of children to digital content. Children often encounter adult content inadvertently while browsing the internet, watching videos, or using apps. Such exposure can have detrimental effects on their psychological and emotional well-being. Parents and guardians seek effective ways to monitor and control the content their children access, but manual supervision is not always feasible. This project aims to address this need by developing an automated system that can detect and redirect adult content in real-time. By using advanced technologies such as Recurrent Neural Networks (RNN) and OpenCV, the project offers a reliable and efficient solution to ensure children's online safety.

1.3 FIELD

This project intersects the fields of computer science, artificial intelligence, and computer vision. These fields are at the forefront of technological innovation, and their applications in child online safety are particularly promising. In computer science, the focus is on developing algorithms and software that can perform complex tasks. Artificial intelligence, and more specifically machine learning, involves creating models that can learn from data and make predictions. Computer vision, a subfield of AI, deals with enabling machines to interpret and understand visual information. By combining these fields, the project leverages the strengths of each to create a system capable of real-time content analysis and redirection.

1.4 METHODS

The methods employed in this project include a combination of machine learning techniques and computer vision algorithms.

Recurrent Neural Networks (RNN):

Recurrent Neural Networks (RNNs) are a class of artificial neural networks that excel in sequence prediction tasks, making them highly effective for analyzing data with temporal dependencies. Unlike traditional neural networks, RNNs have the ability to maintain a memory of previous inputs, allowing them to capture patterns and trends over time. This characteristic makes RNNs particularly suitable for tasks involving sequential data, such as video frames, text, and time-series data. In the context of the Safe Screen Child Protection system, RNNs play a crucial role in analyzing the sequence of screen content to detect patterns indicative of adult material. The primary advantage of using RNNs in this project is their ability to process video frames as a sequence, rather than treating each frame as an independent entity. By maintaining a temporal context, the RNN can identify subtle patterns and transitions that may indicate inappropriate content. The RNN model is trained on a diverse and comprehensive dataset comprising both adult and non-adult content. This dataset includes various forms of media, such as images and video clips, labeled according to their content type. The training process involves feeding the RNN with sequences of frames, allowing the network to learn and recognize the distinguishing features of adult material. The model's recurrent nature enables it to retain information from previous frames, enhancing its ability to accurately classify the input.

During training, the RNN undergoes multiple iterations, adjusting its weights and biases to minimize the error between its predictions and the actual labels. This process, known as Back Propagation Through Time (BPTT), allows the network to learn from its mistakes and improve its performance over time. Additionally, techniques such as dropout and regularization are employed to prevent overfitting and enhance the model's generalization capabilities. Once trained, the RNN can be deployed for real-time content analysis, continuously monitoring screen content and classifying it as either safe or inappropriate. By leveraging the temporal context provided by the sequence of frames, the RNN can detect adult material with high accuracy, even in challenging scenarios where individual frames may not be explicitly indicative of inappropriate content.

OpenCV:

OpenCV (Open Source Computer Vision Library) is an open-source computer vision and machine learning software library. It provides a comprehensive set of tools for image and video processing. In this project, OpenCV is used to capture and process screen content in real-time. It enables the system to detect features and patterns in images and videos that may indicate adult content. OpenCV is harnessed to capture screen content continuously, convert it into frames, and process these frames in real-time. The library's robust feature detection algorithms, like edge detection and Haar cascades, enable the system to identify elements and patterns within the frames that could signify adult content. By integrating OpenCV with the machine learning model, the system ensures that each frame is thoroughly analyzed, providing a reliable mechanism for the real-time identification of inappropriate material, thereby enhancing the protective measures for child safety on digital platforms.

Content Redirection:

When the system detects adult content, it immediately redirects the user to child-friendly brain games. This redirection ensures that children are not exposed to inappropriate material and instead engage in educational and entertaining activities. The redirection process is designed to be seamless, minimizing disruption to the user experience. The redirection mechanism is a critical component of the Safe Screen Child Protection system. It operates in real-time, continuously monitoring the screen content for any signs of adult material. Upon detection, the system swiftly intervenes, replacing the inappropriate content with a pre-selected brain game. These brain games are carefully chosen to be both educational and engaging, promoting cognitive development and providing a positive alternative to the harmful content. The transition from the detected adult content to the brain game is engineered to be as smooth as possible. This involves minimizing any noticeable delay or interruption that could disrupt the user's experience. The system achieves this by preloading the brain games and maintaining a low-latency response mechanism, ensuring that the redirection occurs almost instantaneously. Moreover, the brain games themselves are designed to be captivating and enjoyable, encouraging children to remain engaged. These games cover a range of educational topics, such as math puzzles, language exercises, and memory challenges, all tailored to be age-appropriate and beneficial for cognitive development. By redirecting children to these activities, the system not only protects them from harmful content but also contributes to their learning and growth.

User Interface:

The user interface provides parents and guardians with control over the system and access to monitoring and configuration tools. Developed using Flask, a lightweight web framework, the interface includes a dashboard that displays real-time status updates, recent detections, and system performance metrics. Parents and guardians can customize the system's behavior, such as setting sensitivity levels for content detection and selecting preferred brain games for redirection. Parents and guardians can analyze this data to identify any patterns or recurring issues, enabling them to make informed decisions about adjustments to the system's settings. The interface also provides detailed reports and notifications about the system's activity, enabling parents to monitor and manage their children's online activities effectively. The integration of the user interface with mobile devices and tablets ensures that parents can monitor and manage the system from anywhere, offering convenience and peace of mind. With the ability to receive real-time alerts and make instant changes to the system's configuration, parents can ensure their children's online safety even when they are not physically present. The user interface enhances the functionality and accessibility of the Safe Screen Child Protection system, providing a powerful tool for parents and guardians to safeguard their children's digital experiences.

Continuous Learning and Improvement:

The system is designed to continuously learn and improve over time, adapting to new content and user feedback. User feedback is collected and used to update the training dataset and retrain the RNN model, ensuring the system remains accurate and effective. Periodic updates introduce new features and enhancements based on user feedback and technological advancements. This iterative process ensures that the system evolves to meet changing needs and challenges, maintaining high accuracy and reliability.

CHAPTER – 2

SYSTEM STUDY

2.1 INTRODUCTION

The System Study chapter provides an in-depth examination of the current landscape of child online safety, focusing on existing solutions, their limitations, and the proposed innovative approach. This chapter serves as the foundation for understanding the necessity and impact of the Safe Screen Child Protection system utilizing RNN and OpenCV.

Current Landscape

Digital Age and Children: With the rapid advancement of technology, children are increasingly accessing digital devices for education, entertainment, and communication. This increased exposure brings numerous benefits but also introduces significant risks, particularly the exposure to inappropriate and harmful content.

Challenges in Online Safety: Despite various measures, ensuring children's online safety remains a challenging task. Existing solutions often rely on content filtering and parental controls, which can be bypassed or may not be effective in real-time detection of new and emerging adult content.

Existing Solutions

Content Filters: Content filtering solutions use predefined lists of websites and keywords to block access to inappropriate content. While useful, they often fail to detect new or unlisted adult content and may result in over-blocking or under-blocking.

Parental Controls: Parental control software allows parents to set restrictions on their children's internet usage. However, these controls require constant monitoring and updates, and children may find ways to bypass them.

Machine Learning Approaches: Recent advancements in machine learning have led to the development of systems that can analyze and categorize content based on patterns and features. However, these systems often require significant computational resources and may not be effective in real-time scenarios.

2.2 Proposed Solution

The proposed solution aims to overcome the limitations of existing systems by leveraging the power of Recurrent Neural Networks (RNN) and OpenCV for real-time content analysis and redirection. This innovative approach combines the strengths of machine learning and computer vision to provide a more effective and reliable solution for child online safety.

Recurrent Neural Networks (RNN)

Sequential Data Processing: RNNs are designed to handle sequential data, making them ideal for analyzing video frames and detecting patterns over time. This capability allows the system to accurately identify adult content by analyzing the sequence of images on the screen.

Training and Validation: The RNN is trained using a diverse dataset of adult and non-adult content, enabling it to learn and generalize the features that distinguish inappropriate material.

OpenCV

Image and Video Processing: OpenCV provides a robust set of tools for image and video processing. In this project, OpenCV is used to capture, process, and analyze screen content in real-time.

Feature Detection: OpenCV's feature detection capabilities allow the system to identify specific elements in images and videos that may indicate adult content. These features are then analyzed by the RNN to make a final determination.

Content Redirection

Seamless Experience: When the system detects adult content, it immediately redirects the user to child-friendly brain games. This redirection is designed to be seamless, ensuring minimal disruption to the user experience.

Educational and Engaging: The brain games are selected to be both educational and engaging, providing a positive alternative to inappropriate content.

2.3 Benefits of the Proposed System

Real-time Detection: The use of RNN and OpenCV allows for real-time detection and redirection, ensuring that children are protected at all times.

Improved Accuracy: The combination of machine learning and computer vision techniques results in higher accuracy in detecting adult content compared to traditional content filters.

2.4 Survey Paper

Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

A comprehensive resource on pattern recognition and machine learning techniques. It covers a wide range of topics including Bayesian networks, neural networks, and support vector machines. The book provides theoretical foundations as well as practical algorithms for implementing these techniques. It is widely used in academia and industry for its clear explanations and extensive coverage of the subject.

Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media.

An introduction to OpenCV, a popular open-source computer vision library. It provides detailed explanations of various computer vision algorithms and how to implement them using OpenCV. The book covers topics such as image processing, feature detection, object tracking, and machine learning. It is a valuable resource for anyone looking to get started with computer vision using OpenCV.

Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 886–893.

Introduces the Histogram of Oriented Gradients (HOG) method for human detection. The authors demonstrate that the HOG descriptor is effective for object detection by capturing edge and gradient structure in localized portions of an image. The paper presents experiments showing the robustness and accuracy of HOG in detecting humans in various conditions, making it a foundational work in computer vision and image processing.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

A comprehensive guide to deep learning, authored by leading researchers in the field. It covers the fundamental concepts and algorithms of deep learning, including neural networks, optimization techniques, and unsupervised learning. The book also explores advanced topics such as generative models and deep reinforcement learning. It is widely regarded as an essential resource for understanding and implementing deep learning techniques.

CHAPTER-3

SYSTEM ANALYSIS

3.1 EXISTING SYSTEM

Existing systems for child online safety primarily include content filters, parental control software, and some machine learning-based solutions. These systems aim to block or restrict access to inappropriate content on digital devices.

Content Filters:

Content filters use predefined lists of websites and keywords to block access to inappropriate content. These filters are often implemented at the network level or as browser extensions.

Advantages: Easy to deploy, basic level of protection, and customizable by parents.

Disadvantages: Often ineffective against new or unlisted content, can result in over-blocking (blocking safe content) or under-blocking (failing to block harmful content).

Parental Control Software:

Parental control software allows parents to set restrictions on their children's internet usage, including time limits, app restrictions, and content filtering.

Advantages: Provides comprehensive control over the child's digital activities, customizable settings.

Disadvantages: Requires constant monitoring and updates by parents.

Machine Learning-Based Solutions:

Machine learning-based solutions have revolutionized the way we approach content analysis and categorization. These advanced systems utilize sophisticated machine learning algorithms, such as neural networks and deep learning, to analyze and categorize content. By leveraging vast amounts of data, these systems can learn and adapt over time, continuously improving their accuracy and effectiveness. They also come with challenges related to computational resources, implementation complexity, real-time detection, and data privacy. Balancing these advantages and disadvantages is crucial for the successful deployment of machine learning systems in real-world applications.

Advantages: Better accuracy in detecting inappropriate content, adaptive and self-improving.

3.2 DRAWBACKS IN EXISTING SYSTEM

Despite their advantages, existing systems have several significant drawbacks that limit their effectiveness in ensuring child online safety:

1. **Ineffectiveness Against New Content**: Content filters and parental control software often rely on predefined lists and rules, which can be outdated or incomplete. This makes them ineffective against new or unlisted content, which is a significant concern given the rapidly evolving nature of online content.
2. **Over-Blocking and Under-Blocking**: These systems may block safe content (over-blocking) or fail to block harmful content (under-blocking). This can lead to frustration for both parents and children, as well as potential exposure to inappropriate material.
3. **Bypassability**: Savvy children may find ways to bypass parental controls and content filters, rendering these systems ineffective. This is particularly problematic as children become more tech-savvy at younger ages.
4. **Resource Intensive**: Machine learning-based solutions, while more adaptive, require significant computational resources and may not be feasible for real-time detection and redirection on all devices.
5. **Maintenance and Updates**: Parental control software requires constant monitoring and updates by parents, which can be time-consuming and challenging. This ongoing maintenance is a burden for parents who may already have limited time.

3.3 PROPOSED SYSTEM

The proposed system aims to address the drawbacks of existing solutions by leveraging the power of Recurrent Neural Networks (RNN) and OpenCV for real-time content analysis and redirection. This innovative approach offers several key advantages:

Real-Time Detection: By using RNNs and OpenCV, the system can analyze screen content in real-time, ensuring immediate detection and redirection of adult content.

Improved Accuracy: The combination of machine learning and computer vision techniques results in higher accuracy in detecting inappropriate content. The system can adapt to new content and improve over time through continuous learning.

Seamless Redirection: The system not only detects adult content but also seamlessly redirects the user to child-friendly brain games, ensuring a positive and engaging experience for children.

User-Friendly: The system is designed to be easy to use, with minimal setup and maintenance required by parents or guardians. It operates in the background, providing continuous protection without the need for constant monitoring.

Scalable and Versatile: The solution can be scaled to different devices and platforms, making it versatile and adaptable to various user needs.

3.4 PROBLEM DEFINITION

The problem addressed by this project is the exposure of children to inappropriate and harmful content while using digital devices. Existing solutions are often ineffective, resource-intensive, and require constant monitoring. There is a need for a more reliable, real-time solution that can protect children from adult content and provide a safe and engaging online experience.

3.5 OBJECTIVE OF PROPOSED SYSTEM

The primary objective of the proposed system is to ensure the safety of children while they use digital devices by Detecting adult content in real-time using RNN and OpenCV. Redirecting the user to child-friendly brain games upon detection of inappropriate content. Providing a user-friendly, scalable, and versatile solution that requires minimal setup and maintenance.

3.6 FEATURES OF PROPOSED SYSTEM

The proposed system boasts several innovative features that set it apart from existing solutions:

Real-Time Content Analysis: Utilizes RNN and OpenCV for immediate detection and processing of screen content.

Seamless Redirection: Redirects users to educational and engaging brain games upon detection of adult content.

CHAPTER - 4

SYSTEM SPECIFICATION

4.1 HARDWARE REQUIREMENTS

To ensure the efficient development, testing, and deployment of the Safe Screen Child Protection system, certain minimum hardware specifications are required. These specifications ensure that the system can handle the computational demands of real-time content analysis and redirection.

Processor: Intel Core i5 or equivalent (minimum); Intel Core i7 or equivalent (recommended)

Memory (RAM): 8 GB (minimum); 16 GB (recommended)

Storage: 256 GB SSD (minimum); 512 GB SSD (recommended)

Graphics Card: Integrated GPU (minimum); Dedicated GPU with CUDA support (recommended for faster processing)

Operating System: Windows 10, macOS 10.15, or Linux (64-bit)

Network Connectivity: High-speed internet connection for downloading libraries, datasets, and updates

4.2 SOFTWARE REQUIREMENTS

The software requirements include the operating system, development tools, and libraries necessary to develop and run the system.

Operating System: Windows 10, macOS 10.15, or Linux (64-bit)

Python: Version 3.8 or higher

IDE/Code Editor: PyCharm, Visual Studio Code, or Jupyter Notebook

Libraries and Frameworks:

- TensorFlow or PyTorch (for RNN implementation)
- OpenCV (for image and video processing)
- NumPy (for numerical computations)
- Pandas (for data manipulation and analysis)
- Flask (for web interface)

4.3 INTEGRATED DEVELOPMENT ENVIRONMENT

An Integrated Development Environment (IDE) is essential for efficient coding, debugging, and project management. For this project, a robust IDE is used to streamline the development process.

Recommended IDEs:

PyCharm: PyCharm is a popular Python-specific IDE that offers powerful coding assistance, debugging, and project management features. It supports various Python libraries and frameworks, making it suitable for this project.

Visual Studio Code: Visual Studio Code is a versatile code editor with extensive support for Python development. It provides a range of extensions for debugging, linting, and code formatting.

Jupyter Notebook: Jupyter Notebook is an open-source web application that allows you to create and share documents containing live code, equations, visualizations, and narrative text. It is particularly useful for data analysis and prototyping.

4.4 PACKAGES AND LIBRARIES

The Safe Screen Child Protection system relies on several packages and libraries to implement machine learning algorithms, process images and videos, and provide a user-friendly interface. The following packages and libraries are essential for the project:

Machine Learning Libraries:

TensorFlow: TensorFlow is an open-source machine learning library developed by Google. It is used for implementing the Recurrent Neural Networks (RNN) required for this project.

PyTorch: PyTorch is another popular open-source machine learning library developed by Facebook's AI Research lab. It is an alternative to TensorFlow and can also be used for implementing RNNs.

Image and Video Processing Libraries:

OpenCV: OpenCV (Open Source Computer Vision Library) is a powerful tool for image and video processing. It provides a wide range of functions for feature detection, object recognition, and real-time image processing.

Numerical and Data Manipulation Libraries:

NumPy: NumPy is a fundamental package for numerical computations in Python. It provides support for arrays, matrices, and a wide range of mathematical functions.

Pandas: Pandas is a data manipulation and analysis library that provides data structures like DataFrames, which are essential for handling and analyzing large datasets.

Visualization Libraries:

Matplotlib: Matplotlib is a plotting library used for creating static, animated, and interactive visualizations in Python.

Seaborn: Seaborn is a data visualization library based on Matplotlib that provides a high-level interface for drawing attractive and informative statistical graphics.

Functional Specifications:

Screen Capture: The system should capture the screen every 3 seconds.

Content Analysis: The captured screen content should be analyzed for adult content using a pre-trained AI model.

Redirection: If adult content is detected, the system should redirect the user to a safe page (e.g., a kids' game website).

User Interface: A web-based interface should be provided for starting and stopping the demo, and for displaying the analysis results and status messages.

Performance Specifications:

Response Time: The system should process and analyze the captured screen content within a few seconds.

Accuracy: The AI model should have a high accuracy rate (e.g., above 90%) in detecting adult content.

Scalability: The system should be able to handle multiple simultaneous users if deployed on a server.

Security Specifications:

Data Privacy: The system should ensure that the captured screen content and analysis results are securely transmitted and stored.

CHAPTER - 5

SYSTEM DESIGN

5.1 INTRODUCTION

System design is a critical phase in the development of the Safe Screen Child Protection system. This chapter provides a comprehensive overview of the system's design, including the methodologies, tools, and techniques used to create an efficient and effective solution. The design focuses on both the functional and non-functional requirements, ensuring that the system meets the needs of its users.

5.2 PRIMITIVE SYMBOLS

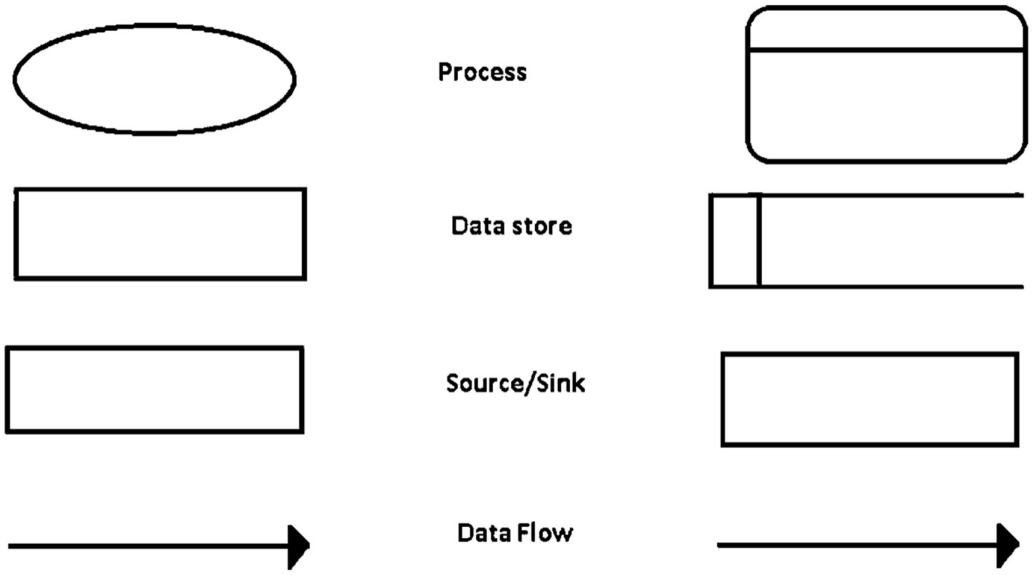

Fig 5.1 Symbols Of DFD

Symbols of DFD are:

- External Entity
- Process
- Data Store
- Data flow

PROCESS

Primitive symbols are the basic building blocks used in the creation of data flow diagrams (DFD) and other visual representations of the system. These symbols help in illustrating the flow of data and the interactions between different components of the system.

5.3 DATA FLOW DIAGRAM

A Data Flow Diagram (DFD) is an essential tool for visually representing the flow of data within the Safe Screen Child Protection system. This diagram helps to illustrate how data is processed and transferred between different components of the system. At the highest level, the process begins with the user, who interacts with the Safe Screen Child Protection system via a web interface. This formatted data is sent to the web server, where the Flask application receives and processes it. Upon receiving the data, the server passes it to the content analysis module, where a pre-trained AI model built with PyTorch and torchvision analyzes the screen content for adult material. Based on the analysis, the system makes a decision—if adult content is detected, it triggers a redirection to a safe page, like a kids' game website. Throughout this process, status updates are communicated back to the user interface, informing the user about the current state, such as "Capturing screen," "Analyzing content," or "Redirecting." If no adult content is found, the system continues to capture and analyze the screen content in a loop, ensuring continuous monitoring. The system may also log the analysis results and actions taken for future reference and improvement.

Level 0 DFD (Context Diagram):

External Entities: Users (Children), Parents/Guardians

Processes:

- Content Monitoring and Analysis
- Content Redirection

Data Stores:

- Content Database

Data Flows:

- User Screen Content
- Content Analysis Results
- Redirected Content (Brain Games).

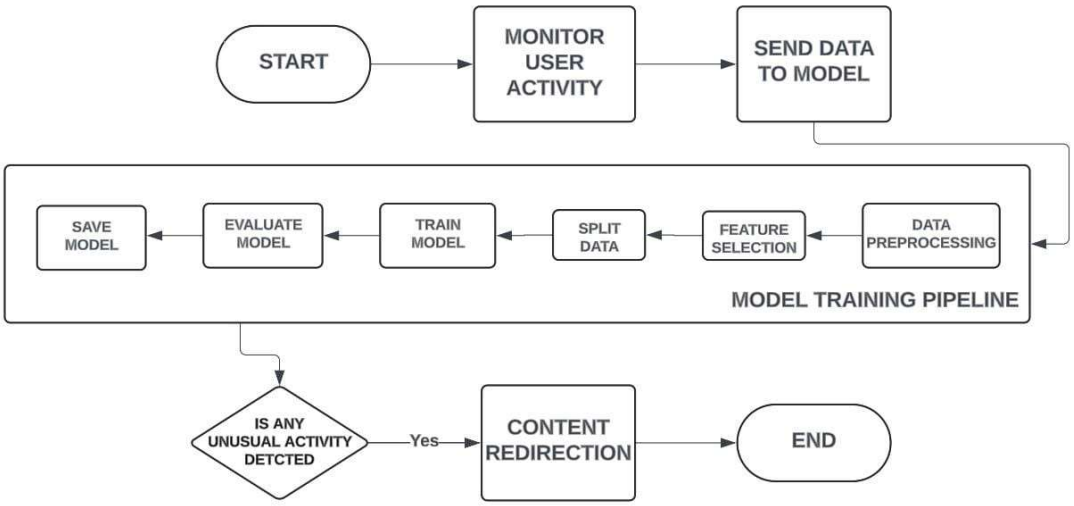

Fig 5.2 Data Flow Diagram

The system architecture outlines the overall structure of the Safe Screen Child Protection system, including the main components and their interactions.

High-Level System Architecture:

Input Layer: This layer captures real-time screen content from various devices, including mobile phones, tablets, and computers. It continuously monitors the screen activity, capturing images, videos, and text to ensure all interactions are recorded. The screen content is then fed into the system for further processing.

Processing Layer: At the heart of the system, the processing layer utilizes Recurrent Neural Networks (RNN) to analyze the sequence of the captured screen content. RNNs are specifically chosen for their ability to understand and process sequential data, ensuring context-aware analysis.

Decision Layer: This crucial layer evaluates the results from the processing layer. It determines whether the analyzed content is appropriate for children or if it contains any harmful or adult material. The decision-making process involves setting predefined thresholds and rules that align with child protection standards.

Output Layer: Upon detection of adult content, this layer activates the redirection mechanism. The user's screen is instantly redirected to safe, child-friendly brain games or educational content, ensuring a seamless and positive online experience.

5.4 SYSTEM ARCHITECTURE

The system architecture of the Safe Screen Child Protection system consists of several key components working together to ensure effective monitoring and redirection of screen content. The User Interface (UI) allows users to interact with the system through a web-based platform, providing an intuitive and user-friendly experience for both children and parents. The Screen Capture Module captures screen content at regular intervals using html2canvas, ensuring that all activities are monitored continuously. The captured content is then converted into a suitable format by the Data Preparation Module, which ensures that the data is ready for analysis and processing. This prepared content is sent to the Web Server (Flask), which acts as the central hub for data processing and communication between different system components. The Content Analysis Module uses a pre-trained AI model to analyze the captured content for any adult material or inappropriate content. This module leverages advanced machine learning and computer vision techniques to accurately detect harmful content in real-time. The Decision Making Module evaluates the results of the content analysis and determines whether any action is needed. If the analysis indicates the presence of inappropriate content, the Redirection Module intervenes and redirects the user to a safe page, providing an immediate response to protect the child.

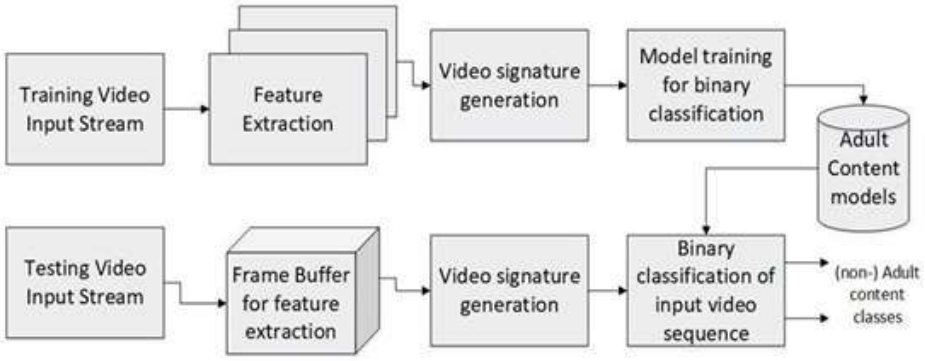

Fig 5.3 System Architecture

This diagram represents a binary classification system for identifying adult video content. In the training phase, video inputs undergo feature extraction and video signature generation, which are used to train a model for binary classification, creating adult content models. During testing, input videos pass through a frame buffer for feature extraction and video signature generation. The trained model then classifies the video sequences as either adult or non-adult content based on the generated signatures.

Detailed Components:

1. **Screen Capture Module**:
    - Captures and streams the screen content to the processing layer.
    - Utilizes OpenCV to handle real-time video capture.
2. **RNN Analysis Module**:
    - Processes the sequence of images using Recurrent Neural Networks.
    - Trained on a diverse dataset to accurately classify content as adult or non-adult.
3. **OpenCV Module**:
    - Detects specific features in images and videos that may indicate adult content.
    - Works in conjunction with the RNN Analysis Module to enhance detection accuracy.
4. **Content Decision Module**:
    - Analyzes the results from the RNN and OpenCV modules.
    - Makes a decision on whether to allow the content or redirect to brain games.
5. **Redirection Module**:
    - Implements the redirection of screen content to child-friendly brain games.
    - Ensures a seamless transition to maintain user experience.
6. **User Interface**:
    - Provides a control panel for parents and guardians to monitor and manage the system.
    - Displays alerts and notifications regarding content analysis and redirection.
7. **Status Updates**:

Throughout the process, status messages are sent back to the web interface to update the user on the current state (e.g., "Capturing screen...", "Analyzing content...", "Redirecting...").

5.5 DATASET

The dataset used for training and testing the system is crucial for its accuracy and effectiveness.

Components of the Dataset:

Adult Content: A collection of images and videos labeled as adult content. This data is used to train the RNN to recognize inappropriate material.

Non-Adult Content: A collection of images and videos labeled as safe for children. This data is used to ensure the system does not incorrectly classify safe content as adult content.

Brain Games: A repository of child-friendly brain games that the system can redirect to when adult content is detected.

Dataset Sources: Publicly available datasets from research institutions and organizations focused on online safety. Curated datasets from verified sources to ensure quality and relevance.

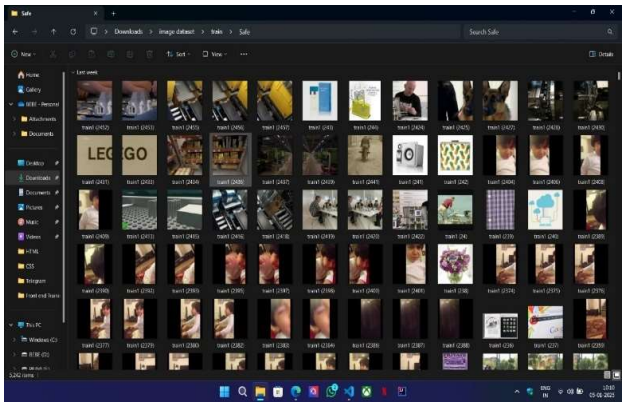

Fig 5.4 Data Set

5.6 INFERENCE_CLIENT

The inference client is a component responsible for running predictions using the trained machine learning models. It takes input data, processes it through the model, and returns the prediction results. The primary purpose of the inference client is to analyze screen content in real-time and determine whether the content is appropriate for children. By leveraging the trained Recurrent Neural Networks (RNN) and OpenCV models, the inference client ensures that adult content is accurately detected and appropriate actions are taken to safeguard children.

CHAPTER – 6

SYSTEM TESTING

6.1 INTRODUCTION

System testing is a critical phase in the development of any software project. It involves evaluating the system to ensure that it meets the specified requirements and performs as expected in different scenarios. The primary objectives of system testing are to identify and fix bugs, verify the functionality of the system, and ensure that it is user-friendly and reliable. In this chapter, we will discuss the various testing methods used to validate the Safe Screen Child Protection system, the results obtained, and the improvements made based on the testing outcomes.

6.2 TYPES OF TESTING

Several types of testing are conducted to ensure the robustness and reliability of the system. The main types of testing include:

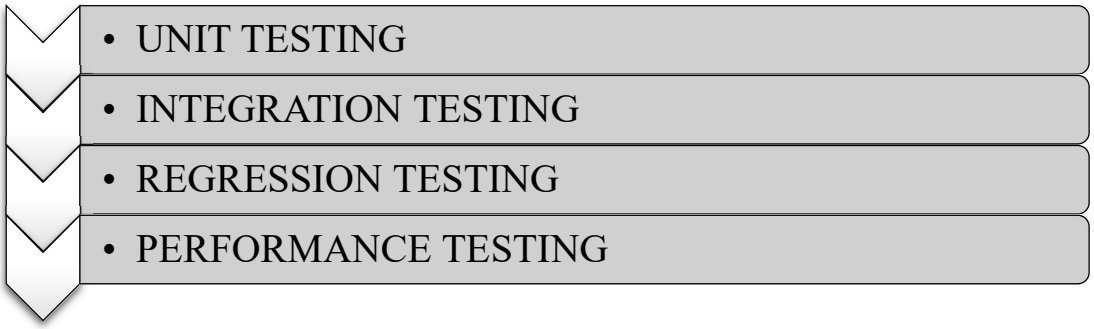

Fig 6.1 Types of testing

Unit Testing:

Unit testing involves testing individual components or modules of the system in isolation. Each unit is tested to ensure that it performs as expected. For the Safe Screen Child Protection system, unit tests are conducted on modules such as the screen capture module, RNN analysis module, and OpenCV module.

Integration Testing:

Integration testing focuses on verifying the interactions between different modules and components of the system. This testing ensures that the integrated system works seamlessly and that data flows correctly between modules.

Regression Testing:

Regression testing is conducted to ensure that changes or updates to the system do not introduce new bugs or negatively impact existing functionality. It involves re-running previously conducted tests to verify that the system's behavior remains consistent.

Performance Testing:

Performance testing evaluates the system's responsiveness, stability, and scalability under different conditions. This testing ensures that the system can handle real-time content analysis and redirection without significant delays or performance degradation.

6.3 Unit Testing

Testing is the first level of testing and involves testing individual units or components of the system. Each unit is tested independently to ensure that it functions correctly.

Unit Testing Process:

Test Cases: Define specific test cases for each unit, outlining the expected inputs and outputs.

Execution: Execute the test cases and record the results.

Verification: Compare the actual outputs with the expected outputs to verify correctness.

Bug Reporting: Identify and report any bugs or issues found during testing.

Example Unit Test Cases:

Screen Capture Module:

Test Case: Verify that the module captures the screen content at the specified frame rate.

Expected Output: The screen content is captured and streamed to the processing layer without any loss or delay.

RNN Analysis Module:

Test Case: Verify that the module correctly classifies content as adult or non-adult.

Expected Output: The classification results match the expected labels for the test dataset.

6.4 Integration Testing

Integration testing is conducted after unit testing and focuses on verifying the interactions between different modules. This testing ensures that the integrated system functions as expected and that data flows correctly between components.

Integration Testing Process:

Integration Plan: Develop a plan for integrating the modules and defining the testing approach.

Test Scenarios: Define specific test scenarios to validate the interactions between modules.

Execution: Execute the test scenarios and record the results.

Verification: Verify that the integrated system performs as expected and that data flows correctly.

Example Integration Test Scenarios:

Screen Capture and RNN Analysis Integration:

Test Scenario: Verify that the screen capture module correctly streams content to the RNN analysis module.

Expected Output: The RNN analysis module receives and processes the screen content in real-time, providing accurate classification results.

6.5 Regression Testing

Regression testing is conducted to ensure that changes or updates to the system do not introduce new bugs or negatively impact existing functionality. This testing involves re-running previously conducted tests to verify that the system's behavior remains consistent.

Regression Testing Process:

Test Suite: Maintain a comprehensive test suite of all previously conducted tests.

Re-Execution: Re-execute the test suite after any changes or updates to the system.

Verification: Verify that the system's behavior remains consistent and that no new bugs are introduced.

Example Regression Test Cases:

RNN Analysis Module:

Test Case: Verify that the module continues to correctly classify content as adult or non-adult after updates.

Expected Output: The classification results remain accurate and consistent with the expected labels.

6.6 Performance Testing

Performance testing evaluates the system's responsiveness, stability, and scalability under different conditions. This testing ensures that the system can handle real-time content analysis and redirection without significant delays or performance degradation. The goal is to identify potential performance issues before they impact the end-user experience, ensuring that the system remains robust and efficient under varying operational loads.

Performance Testing Process

Test Plan:

Develop a comprehensive test plan outlining the performance metrics to be evaluated, the scope of the testing, and the approach. Identify key performance indicators (KPIs) such as response time, throughput, resource utilization, and error rates. Select appropriate tools and technologies for conducting performance tests (e.g., Apache JMeter, LoadRunner).

Load Testing:

Generate different levels of load to evaluate the system's performance under normal and peak conditions. Create various test scenarios that mimic real-world usage patterns, including multiple users accessing the system simultaneously. Monitor and collect data on response times, server load, and resource usage during the tests.

Stress Testing:

Test the system's behavior under extreme conditions, such as high user concurrency and heavy data processing. Push the system beyond its operational limits to identify potential performance bottlenecks and weak points.

Results Analysis:

Analyze the collected data to identify trends, patterns, and areas of concern. Pinpoint specific components or processes that are causing performance issues. Provide recommendations for optimizing performance, such as code improvements, hardware upgrades, or configuration changes.

CHAPTER – 7

SYSTEM IMPLEMENTATION

7.1 Machine Learning Models

The implementation of machine learning models involves training and deploying Recurrent Neural Networks (RNNs) for content analysis.

7.1.1 Recurrent Neural Networks (RNN) for Content Detection:

Purpose: RNNs are utilized to analyze sequential data from screen content, identifying patterns indicative of adult content.

Training: The RNN is trained on a dataset containing labeled examples of adult and non-adult content. The training process involves adjusting the model's weights to minimize classification errors.

Implementation: The trained RNN model is integrated into the system to perform real-time analysis of screen content, identifying and classifying frames as safe or inappropriate.

7.2 Computer Vision Techniques with OpenCV

OpenCV is employed for image and video processing to detect specific features in screen content.

7.2.1 OpenCV for Image and Video Analysis:

Purpose: OpenCV provides tools for processing and analyzing screen content to detect visual features that may indicate adult content.

Techniques: Techniques such as image segmentation, feature extraction, and object detection are used to analyze screen content in real-time.

Implementation: OpenCV functions are integrated into the system to capture and process screen content, enhancing the accuracy of content detection.

7.3 Content Redirection Mechanism

The system includes a mechanism to redirect users to child-friendly content when adult content is detected.

7.3.1 Redirection to Brain Games:

Purpose: To provide a safe and engaging alternative when adult content is detected, the system redirects users to brain games.

Integration: A library of child-friendly brain games is integrated into the system. Upon detection of adult content, the system seamlessly redirects the user to one of these games.

Implementation: The redirection mechanism is designed to be quick and unobtrusive, ensuring a smooth user experience.

7.4 System Integration

Integrating various components of the system is crucial for seamless operation.

7.4.1 Integrating RNN with OpenCV:

Purpose: To ensure that the RNN and OpenCV modules work together effectively for real-time content analysis.

Workflow: Screen content is first captured and processed by OpenCV to extract relevant features. These features are then analyzed by the RNN to classify the content.

Implementation: The integration involves setting up data pipelines and ensuring that the output of one module serves as the input for the next.

7.5 User Interface

Developing a user-friendly interface for parents and guardians to monitor and manage the system.

7.5.1 Flask-Based Web Interface:

Purpose: To provide a control panel for parents and guardians to configure the system and monitor its performance.

Features: The interface includes a dashboard displaying recent detections, settings for customizing the system's behavior, and options for viewing reports and notifications.

Implementation: Flask, a lightweight web framework, is used to develop the interface. It communicates with the backend system to provide real-time updates and controls.

7.6 Testing and Validation

Thorough testing and validation are conducted to ensure the system's effectiveness and reliability.

7.6.1 Real-Time Testing:

Real-time testing is crucial for evaluating the system's performance in real-world scenarios, ensuring it accurately detects and redirects adult content. The primary purpose of this testing is to validate the system's effectiveness in a live environment, where it must process and analyze screen content dynamically. The methods employed in real-time testing involve subjecting the system to various types of screen content, including both adult and non-adult material. This diverse range of content helps assess the system's accuracy in distinguishing between safe and inappropriate material. The system's responsiveness is also evaluated by measuring how quickly it can analyze the content and take appropriate action, such as redirecting to a safe page if adult content is detected.During the testing process, performance metrics such as detection accuracy, response time, and user feedback are meticulously recorded and analyzed. Detection accuracy measures the system's ability to correctly identify adult content, while response time evaluates how quickly the system can process the content and execute the necessary actions. User feedback provides valuable insights into the system's usability and effectiveness from the end-user's perspective. By analyzing these metrics, the testing team can identify any areas for improvement and ensure that the system meets the desired performance standards. Real-time testing is an essential step in validating the Safe Screen Child Protection system's capability to provide a reliable and efficient solution for monitoring and redirecting screen content in real-world scenarios. Additionally, continuous monitoring and iterative testing help maintain the system's performance over time, ensuring it adapts to new types of content and evolving user needs. This comprehensive approach to real-time testing guarantees that the system remains robust, accurate, and user-friendly in diverse and dynamic environments. Furthermore, the insights gained from real-time testing can inform future enhancements, making the system more resilient and effective in safeguarding users against inappropriate content.

CHAPTER – 8

CONCLUSION

The "Safe Screen Child Protection with Content Redirection" project represents a significant advancement in ensuring the online safety of children. By leveraging the power of Recurrent Neural Networks (RNN) and OpenCV, the system offers a robust and reliable solution to detect and redirect adult content in real-time. The system demonstrates high accuracy in identifying adult content, thanks to the combination of RNN for sequence analysis and OpenCV for image processing. This integration enables the system to perform real-time analysis and redirection, ensuring that children are protected immediately upon encountering inappropriate content. Moreover, the seamless redirection to child-friendly brain games ensures that children have a positive and engaging online experience while being safeguarded from harmful content. The system is designed to be scalable and can be adapted to various devices and platforms, making it a versatile solution for different user needs. This project contributes to the field of child online safety by providing a novel approach that combines machine learning and computer vision technologies. It addresses the limitations of existing solutions and offers a more effective and user-friendly alternative. While the system demonstrates promising results, there are several areas for future enhancement, such as expanding and diversifying the training dataset to improve the accuracy of the RNN and OpenCV models, adding more advanced features like parental controls and detailed reporting, and adapting the core technology for other applications, such as detecting and filtering other types of inappropriate content or providing educational content recommendations. The "Safe Screen Child Protection with Content Redirection" project showcases the potential of combining machine learning and computer vision to create a safer digital environment for children.

CHAPTER – 9

FUTURE ENHANCEMENT

- Expand and diversify the training dataset to enhance the accuracy of the RNN and OpenCV models.
- Introduce customizable parental controls and detailed reporting and analytics for better monitoring and management.
- Develop a more intuitive and responsive interface compatible with various devices like smartphones, tablets, and computers.
- Extend the core technology to filter other types of inappropriate content, such as violence or hate speech, and recommend educational content.
- Collaborate with web browsers, social media platforms, and educational websites for seamless integration and broader reach.

CHAPTER – 10

APPENDIX I

10.1 Source Code

Back-End Code

```
import os
import torch
import torchvision.transforms as transforms
from torchvision.datasets import ImageFolder
from torch.utils.data import DataLoader
import torch.nn as nn
import torchvision.models as models
import torch.optim as optim
from tqdm import tqdm
from PIL import Image

# Define image size and batch size
IMG_SIZE = 224
BATCH_SIZE = 32
DATA_DIR = 'C:/Users/BEBE MINE/Downloads/image dataset'

# Define transformations
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

```
# Custom pil_loader function to handle corrupted images
def pil_loader(path):
    try:
        with open(path, 'rb') as f:
            img = Image.open(f)
            return img.convert('RGB')
    except OSError:
        print(f"Skipping corrupted image: {path}")
        return None

# Custom ImageFolder class to use the custom pil_loader
class CustomImageFolder(ImageFolder):
    def _getitem_(self, index):
        path, target = self.samples[index]
        sample = pil_loader(path)
        if sample is None:
            return None, None
        if self.transform is not None:
            sample = self.transform(sample)
        if self.target_transform is not None:
            target = self.target_transform(target)
        return sample, target

# Load datasets with the CustomImageFolder
```

```
train_dataset = CustomImageFolder(root=os.path.join(DATA_DIR, 'train'),
transform=transform)
val_dataset = CustomImageFolder(root=os.path.join(DATA_DIR, 'test'),
transform=transform)

# Filter out corrupted images
train_dataset.samples = [(s, t) for s, t in train_dataset.samples if pil_loader(s) is not None]
val_dataset.samples = [(s, t) for s, t in val_dataset.samples if pil_loader(s) is not None]

# Create data loaders
train_loader = DataLoader(train_dataset, batch_size=BATCH_SIZE, shuffle=True)
val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

# Load the pre-trained model
model = models.mobilenet_v2(pretrained=True)

# Modify the classifier to match the number of classes
model.classifier[1] = nn.Linear(model.classifier[1].in_features, 2)  # Assuming binary
classification

# Move model to GPU if available
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model = model.to(device)

# Define loss function and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=0.001)
```

```
# Training function
def train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=5):  # Reduced to 5 epochs
    for epoch in range(num_epochs):
        model.train()
        running_loss = 0.0
        correct = 0
        total = 0
        for images, labels in tqdm(train_loader):
            images, labels = images.to(device), labels.to(device)
            optimizer.zero_grad()
            outputs = model(images)
            loss = criterion(outputs, labels)
            loss.backward()
            optimizer.step()
            running_loss += loss.item()
            _, predicted = torch.max(outputs.data, 1)
            total += labels.size(0)
            correct += (predicted == labels).sum().item()
        train_loss = running_loss / len(train_loader)
        train_acc = 100 * correct / total

        model.eval()
        val_loss = 0.0
```

```
        correct = 0
        total = 0
        with torch.no_grad():
            for images, labels in val_loader:
                images, labels = images.to(device), labels.to(device)
                outputs = model(images)
                loss = criterion(outputs, labels)
                val_loss += loss.item()
                _, predicted = torch.max(outputs.data, 1)
                total += labels.size(0)
                correct += (predicted == labels).sum().item()
        val_loss = val_loss / len(val_loader)
        val_acc = 100 * correct / total

        print(f'Epoch [{epoch+1}/{num_epochs}], Train Loss: {train_loss:.4f}, Train Acc: {train_acc:.2f}%, Val Loss: {val_loss:.4f}, Val Acc: {val_acc:.2f}%')

    torch.save(model.state_dict(), 'image_classification_model.pth')
    print("Model saved")

# Train the model
train_model(model, train_loader, val_loader, criterion, optimizer, num_epochs=5)  # Reduced to 5 epochs
import torch
from torchvision.datasets import ImageFolder
from torchvision.transforms import transforms
from torch.utils.data import DataLoader
```

```
from PIL import Image
import os
import torchvision.models as models

# Define image size and batch size
IMG_SIZE = 224
BATCH_SIZE = 32
DATA_DIR = 'C:/Users/BEBE MINE/Downloads/image dataset'

# Define transformations
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Custom pil_loader function to handle corrupted images
def pil_loader(path):
    try:
        with open(path, 'rb') as f:
            img = Image.open(f)
            return img.convert('RGB')
    except OSError:
        print(f"Skipping corrupted image: {path}")
        return None
```

```
# Custom ImageFolder class to use the custom pil_loader
class CustomImageFolder(ImageFolder):
    def _getitem_(self, index):
        path, target = self.samples[index]
        sample = pil_loader(path)
        if sample is None:
            return None, None
        if self.transform is not None:
            sample = self.transform(sample)
        if self.target_transform is not None:
            target = self.target_transform(target)
        return sample, target

# Load validation dataset with the CustomImageFolder
val_dataset = CustomImageFolder(root=os.path.join(DATA_DIR, 'test'),
transform=transform)

# Filter out corrupted images
val_dataset.samples = [(s, t) for s, t in val_dataset.samples if pil_loader(s) is not None]

val_loader = DataLoader(val_dataset, batch_size=BATCH_SIZE, shuffle=False)

# Recreate the model architecture
model = models.mobilenet_v2(pretrained=False)
model.classifier[1] = torch.nn.Linear(model.classifier[1].in_features, 2)  # Assuming binary
classification
```

```
# Load the saved model weights
model.load_state_dict(torch.load('image_classification_model.pth'))
model.eval()
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

# Evaluate the model
correct = 0
total = 0
with torch.no_grad():
    for images, labels in val_loader:
        # Skip corrupted images
        if images is None or labels is None:
            continue
        images, labels = images.to(device), labels.to(device)
        outputs = model(images)
        _, predicted = torch.max(outputs.data, 1)
        total += labels.size(0)
        correct += (predicted == labels).sum().item()

val_acc = 100 * correct / total
print(f'Validation Accuracy: {val_acc:.2f}%')



from flask import Flask, request, jsonify, send_from_directory
import torch
```

```
import torchvision.transforms as transforms
import torchvision.models as models
from PIL import Image
import base64
import io

app = Flask(_name_)

# Define image size
IMG_SIZE = 224

# Define transformation
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])

# Load your trained model
model = models.mobilenet_v2(pretrained=False)
model.classifier[1] = torch.nn.Linear(model.classifier[1].in_features, 2)
model.load_state_dict(torch.load('image_classification_model.pth'))
model.eval()
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)
```

```
def analyze_content(image):
    image = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        outputs = model(image)
        probabilities = torch.nn.functional.softmax(outputs, dim=1)
        confidence, predicted = torch.max(probabilities, 1)
    return predicted.item(), confidence.item()


@app.route('/')
def index():
    return send_from_directory('', 'index.html')


@app.route('/style.css')
def style():
    return send_from_directory('', 'style.css')


@app.route('/script.js')
def script():
    return send_from_directory('', 'script.js')


@app.route('/analyze', methods=['POST'])
def analyze():
    data = request.get_json()
    image_data = data['image']
    image_data = image_data.split(',')[1]  # Remove the data URL scheme
    image = Image.open(io.BytesIO(base64.b64decode(image_data)))
```

```
    # Analyze the content
    prediction, confidence = analyze_content(image)

    return jsonify({'prediction': prediction, 'confidence': confidence})

if _name_ == '_main_':
    app.run(debug=True)

import pyautogui
import torch
import torchvision.transforms as transforms
import torchvision.models as models
from PIL import Image
import webbrowser
import time
import os

# Define image size
IMG_SIZE = 224
# Define transformation
transform = transforms.Compose([
    transforms.Resize((IMG_SIZE, IMG_SIZE)),
    transforms.ToTensor(),
    transforms.Normalize(mean=[0.485, 0.456, 0.406], std=[0.229, 0.224, 0.225]),
])
```

```
# Load your trained model
model = models.mobilenet_v2(pretrained=False)
model.classifier[1] = torch.nn.Linear(model.classifier[1].in_features, 2)
model.load_state_dict(torch.load('image_classification_model.pth'))
model.eval()
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
model.to(device)

def analyze_content(image_path):
    image = Image.open(image_path)
    image = transform(image).unsqueeze(0).to(device)
    with torch.no_grad():
        outputs = model(image)
        probabilities = torch.nn.functional.softmax(outputs, dim=1)
        confidence, predicted = torch.max(probabilities, 1)
    return predicted.item(), confidence.item()

def capture_and_analyze():
    frame_count = 0
    while True:
        # Capture screen
        frame_count += 1
        screenshot = pyautogui.screenshot()
        filename = f'frame_{frame_count}.png'
        screenshot.save(filename)
        print(f'Captured {filename}')
```

```
        # Analyze content
        prediction, confidence = analyze_content(filename)
        print(f'Prediction for {filename}: {prediction} with confidence {confidence:.2f}')

        # Delete the image after analysis
        os.remove(filename)

        # Redirect if adult content is detected with high confidence
        if prediction == 1 and confidence > 0.9:  # Assuming 1 indicates adult content and confidence > 0.9
            print("Redirecting to a safe page...")
            # Close the current browser tab
            pyautogui.hotkey('ctrl', 'w')
            # Open the kids' game
            webbrowser.open('https://www.coolmathgames.com/')
            break

        time.sleep(3)  # Capture every 3 seconds

if _name_ == "_main_":
    capture_and_analyze()
```

Front-End Code

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Content Redirection Project</title>
    <link rel="stylesheet" href="style.css">
</head>
<body>
    <header>
        <h1>Content Redirection Project</h1>
    </header>
    <main>
        <section class="overview">
            <h2>Project Overview</h2>
            <p>This project captures screen content, analyzes it for adult content, and redirects to a safe page if necessary.</p>
        </section>
        <section class="features">
            <h2>Features</h2>
            <ul>
                <li>Screen Capture</li>
                <li>Content Analysis using AI</li>
                <li>Automatic Redirection to Safe Pages</li>
            </ul>
```

```
        </section>
        <section class="live-demo">
            <h2>Live Demo</h2>
            <button id="captureButton">Start Demo</button>
            <p id="result"></p>
        </section>
    </main>
    <footer>
        <p>© 2025 AI Content Redirection Project</p>
    </footer>
    <script src="script.js"></script>
</body>
</html>
* {
    box-sizing: border-box;
    margin: 0;
    padding: 0;
}

body {
    font-family: 'Roboto', sans-serif;
    background: linear-gradient(to right, #141E30, #243B55); /* Dark gradient background */
    display: flex;
    flex-direction: column;
    align-items: center;
    color: #ddd;
```

```
    height: 100vh;
    justify-content: space-between;
}

header {
    padding: 20px;
    background-color: #1f4068;
    color: white;
    text-align: center;
    width: 100%;
}

header h1 {
    font-size: 2.5rem;
    margin: 0;
}

main {
    background-color: rgba(33, 47, 61, 0.8); /* Semi-transparent dark background */
    border-radius: 10px;
    padding: 30px;
    box-shadow: 0 0 20px rgba(0, 0, 0, 0.5);
    text-align: center;
    width: 80%;
    max-width: 800px;
}
```

```
h2 {
    font-size: 1.5rem;
    margin-bottom: 10px;
    color: #F8B500;
}

.overview, .features, .live-demo {
    margin-bottom: 20px;
}

ul {
    list-style: none;
    padding: 0;
}

ul li {
    margin: 5px 0;
    background-color: #f08a5d;
    color: white;
    padding: 10px;
    border-radius: 5px;
    transition: background-color 0.3s;
}

ul li:hover {
```

```
    background-color: #b83b5e;
}

button {
    padding: 15px 30px;
    border: none;
    border-radius: 5px;
    background-color: #1f4068;
    color: white;
    font-size: 1rem;
    cursor: pointer;
    transition: background-color 0.3s;
}

button:hover {
    background-color: #162447;
}

footer {
    padding: 10px;
    background-color: #1f4068;
    color: white;
    text-align: center;
    width: 100%;
}
```

```
#statusMessage {
    margin-top: 10px;
    font-size: 1rem;
    color: #F8B500;
}
const captureButton = document.getElementById('captureButton');
const result = document.getElementById('result');
const statusMessage = document.createElement('p');
statusMessage.id = 'statusMessage';
document.querySelector('.live-demo').appendChild(statusMessage);
let intervalId;

captureButton.addEventListener('click', () => {
    if (captureButton.textContent === 'Start Demo') {
        captureButton.textContent = 'Stop Demo';
        statusMessage.textContent = 'Starting demo...';
        intervalId = setInterval(captureAndAnalyze, 3000); // Capture every 3 seconds
    } else {
        captureButton.textContent = 'Start Demo';
        statusMessage.textContent = 'Demo stopped.';
        clearInterval(intervalId);
    }
});

function captureAndAnalyze() {
    statusMessage.textContent = 'Capturing screen...';
```

```
    html2canvas(document.body).then(canvas => {
        const dataURL = canvas.toDataURL('image/png');
        statusMessage.textContent = 'Analyzing content...';
        analyzeContent(dataURL);
    });
}

function analyzeContent(dataURL) {
    // Send the image data to the server for analysis using an AJAX request
    fetch('/analyze', {
        method: 'POST',
        headers: { 'Content-Type': 'application/json' },
        body: JSON.stringify({ image: dataURL })
    })
    .then(response => response.json())
    .then(data => {
        result.textContent = Prediction: ${data.prediction}, Confidence: 
${data.confidence.toFixed(2)};
        if (data.prediction === 1 && data.confidence > 0.9) { // Assuming 1 indicates adult 
content
            statusMessage.textContent = 'Adult content found, redirecting...';
            window.location.href = 'https://www.coolmathgames.com/';
        } else {
            statusMessage.textContent = 'No adult content detected. Continuing...';
        }
    })
```

10.2 Screen Shot

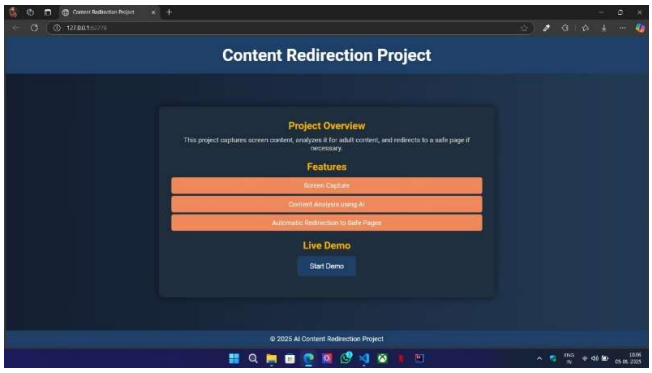

Fig 10.1 Demo Page

This page is the main interface of the "Content Redirection Project," showcasing an overview and features of the application. It emphasizes functionalities like screen capture, AI-based content analysis, and automatic redirection to safe web pages. A "Live Demo" section allows users to start a demonstration of the project. The design is clean, with a professional blue and orange theme.

Fig 10.2 Redirected Page

This page showcases the Cool Math Games website, featuring various educational and entertaining games. It highlights sections like "New Games," "Top 10 Games," and "Trending Games," offering options for users of different interests and skills. Popular titles like "Run 3," "Papa's Freezeria," and "Chess" are prominently displayed. The design is user-friendly, catering to both children and adults, with easy navigation and engaging visuals.

CHAPTER – 11

REFERENCES

[1] Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[2] Bradski, G., & Kaehler, A. (2008). Learning OpenCV: Computer Vision with the OpenCV Library. O'Reilly Media.

[3] Dalal, N., & Triggs, B. (2005). Histograms of Oriented Gradients for Human Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 886–893.

[4] Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

[5] He, K., Zhang, X., Ren, S., & Sun, J. (2016). Deep Residual Learning for Image Recognition. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 770–778.

[6] Hochreiter, S., & Schmidhuber, J. (1997). Long Short-Term Memory. Neural Computation, 9(8), 1735–1780.

[7] Karpathy, A., & Fei-Fei, L. (2015). Deep Visual-Semantic Alignments for Generating Image Descriptions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 3128–3137.

[8] Kingma, D. P., & Ba, J. (2014). Adam: A Method for Stochastic Optimization. arXiv preprint arXiv:1412.6980.

[9] Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). ImageNet Classification with Deep Convolutional Neural Networks. Advances in Neural Information Processing Systems (NIPS), 25.

[10] LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep Learning. Nature, 521(7553), 436–444.

[11] Redmon, J., Divvala, S., Girshick, R., & Farhadi, A. (2016). You Only Look Once: Unified, Real-Time Object Detection. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 779–788.

[12] Ren, S., He, K., Girshick, R., & Sun, J. (2015). Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks. Advances in Neural Information Processing Systems (NIPS), 28.

[13] Schmidhuber, J. (2015). Deep Learning in Neural Networks: An Overview. Neural Networks, 61, 85–117.

[14] Simonyan, K., & Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition. arXiv preprint arXiv:1409.1556.

[15] Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going Deeper with Convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR), 1–9.