1. INTRODUCTION

1.1 ABSTRACT

The Dental Problem Detection App represents a pioneering venture into the realm of healthcare technology, specifically targeting the early identification of dental issues using deep learning and mobile applications. This innovative project harnesses the power of Convolutional Neural Networks (CNN) to analyze X-ray images for the detection of conditions such as tooth decay, impacted teeth, and abscesses.

The primary objective of the Dental Problem Detection App is to provide a user-friendly and accessible tool for individuals to assess their dental health through the analysis of X-ray images. By leveraging deep learning techniques, the app offers real-time classification of dental issues, empowering users with immediate insights into potential problems.

In this project, a CNN model, trained on a diverse dataset sourced from Kaggle, serves as the core engine for dental issue identification. The integration of this model into an Android app, developed using Android Studio, enables users to capture X-ray images via the camera or select from the gallery. The app then processes these images, delivering instant results on the presence of tooth decay, impacted teeth, or abscesses.

The choice of the VGG (Visual Geometry Group) model architecture for the CNN underscores the commitment to extracting intricate features from X-ray images, ensuring a high level of accuracy in classification. The project not only addresses the critical need for early detection of dental problems but also serves as a technological advancement in promoting oral health awareness.

With a user-centric approach, the Dental Problem Detection App offers a seamless interface for image capture, analysis, and result presentation. The prevention information associated with each classified dental issue enhances the app's utility by providing users with actionable insights to maintain optimal oral health.

This project is not just a technological endeavor; it represents a significant step toward democratizing healthcare by placing diagnostic tools directly in the hands of individuals. The fusion of deep learning and mobile technology exemplifies the potential for transformative advancements in the healthcare landscape.

As the field of dental diagnostics evolves, the Dental Problem Detection App stands as a testament to the possibilities of leveraging emerging technologies for the benefit of public health. The convergence of deep learning, mobile applications, and dental care marks a new frontier in accessible and proactive healthcare solutions.

1.2 PROJECT DESCRIPTION

The Dental Problem Detection App project is an innovative foray into the intersection of healthcare and technology, specifically designed to revolutionize dental diagnostics through the integration of deep learning and mobile applications.

Objective:

- The main objective is to create a user-friendly and accessible app that employs Convolutional Neural Networks (CNN) to analyze X-ray images, enabling the early identification of dental issues such as tooth decay, impacted teeth, and abscesses.

Key Components:

1. Deep Learning Model: A CNN model serves as the backbone of the project, trained on a diverse dataset obtained from Kaggle. This model plays a pivotal role in accurately classifying dental conditions based on X-ray images.

2. Android Application: The deep learning model is seamlessly integrated into an Android app developed using Android Studio. The app is designed to run on devices with Android version 7.0 and above, ensuring widespread compatibility.

3. Image Capture and Processing: The app allows users to capture X-ray images using the device's camera or select images from the gallery. These images are then processed using the trained CNN model to identify dental problems.

4. Prevention Information: For each identified dental issue, the app provides relevant prevention information, empowering users with actionable insights to maintain optimal oral health.

Technological Stack:

- Python and TensorFlow: The deep learning model is trained using Python, leveraging the TensorFlow framework for efficient model development and training.

- **Android Studio:** The Android application is developed using Android Studio, a powerful integrated development environment (IDE) for Android app development.

Workflow:

1. Data Collection: A diverse dataset is sourced from Kaggle, comprising X-ray images representing different dental conditions.

2. Model Training: The CNN model is trained using Python in the IDLE environment, employing TensorFlow for efficient neural network training.

3. Android App Development: The trained model is integrated into an Android app developed in Android Studio. The app allows users to capture or select X-ray images for analysis.

4. Real-time Analysis: The app processes the captured or selected images using the CNN model, providing real-time identification of dental issues.

5. User Interaction: The app displays the classified dental issue and offers prevention information, enhancing user awareness and promoting proactive oral healthcare.

Significance:

- The Dental Problem Detection App represents a crucial advancement in dental diagnostics, democratizing access to early identification of dental issues. By placing diagnostic tools directly in the hands of users, the project contributes to the broader landscape of accessible and proactive healthcare solutions.

Future Enhancements:

- Future enhancements may include expanding the dataset for more comprehensive training, refining the CNN model for increased accuracy, and incorporating additional features for a holistic dental health assessment.

In essence, the Dental Problem Detection App project signifies a fusion of cutting-edge technology and healthcare, with the potential to transform how individuals engage with their oral health. The project's impact extends beyond mere diagnostics, fostering a culture of preventive healthcare through technological innovation.

2. SYSTEM ANALYSIS

2.1 EXISTING SYSTEM AND ITS DISADVANTAGES

In the realm of dental diagnostics, the traditional approaches lack the dynamic capabilities and user-centric features offered by the Dental Problem Detection App. The existing system, often reliant on manual diagnostic methods, is characterized by its limitations and drawbacks.

DISADVANTAGES:

1. Manual Programming Dependency:

- In the prevailing system, diagnostic procedures are commonly programmed with a fixed set of instructions. This rigid structure limits adaptability to evolving dental diagnostic needs.

2. Vulnerability to Program Crashes:

- Changes or updates to the program can introduce unforeseen issues, potentially leading to system crashes. The lack of flexibility in accommodating modifications hinders the agility required for effective dental diagnostics.

3. Limited User-Friendliness:

- The existing systems typically lack user-friendly interfaces, making it challenging for both healthcare professionals and users to interact seamlessly. Complex procedures and non-intuitive designs may impede effective utilization.

4. Reduced Accessibility:

- Accessibility to dental diagnostic tools in the traditional system is often restricted to clinical settings. This limitation hampers the widespread availability of diagnostic capabilities, particularly in remote or non-clinical environments.

5. Scalability Challenges:

- Scaling up the diagnostic capabilities or incorporating new features in the existing system is often a cumbersome process. The lack of scalability can hinder advancements in dental diagnostics technology.

6. Limited Real-time Interaction:

- Traditional diagnostic methods may not provide real-time feedback and interaction. This limitation can delay the identification of dental issues and the implementation of preventive measures.

The Dental Problem Detection App addresses these disadvantages by introducing a dynamic, user-friendly, and technologically advanced approach to dental diagnostics. Through the integration of deep learning and mobile technology, the app overcomes the constraints of the existing system, offering a more versatile and accessible solution for both healthcare professionals and users.

2.2 PROPOSED SYSTEM AND ITS ADVANTAGES

The proposed Dental Problem Detection App represents a paradigm shift from the constraints of the existing systems, introducing a dynamic and user-centric approach to dental diagnostics.

ADVANTAGES:

1. Flexibility in Device Control:

- Unlike the rigid programming of the existing system, the proposed system empowers users to change device statuses effortlessly. This flexibility allows users to interact with the app intuitively, enhancing the overall diagnostic experience.

2. Efficiency in Dental Diagnostics:

- The proposed system leverages deep learning and mobile technology, enabling efficient and real-time dental diagnostics. Users can swiftly capture or select X-ray images, and the app, utilizing a trained CNN model, identifies dental problems promptly, fostering a more streamlined diagnostic process.

3. Low-Cost Management:

- By utilizing the capabilities of smartphones and integrating deep learning on a mobile platform, the proposed system eliminates the need for expensive specialized diagnostic

equipment. This cost-effective approach democratizes access to dental diagnostics, particularly in resource-constrained settings.

4. Enhanced User-Friendliness:

- The user interface of the Dental Problem Detection App is designed to be intuitive and user-friendly. With seamless navigation, image capture options, and prevention information display, users can engage with the app effortlessly, promoting proactive dental healthcare.

5. Accessibility Beyond Clinical Settings:

- The proposed system extends accessibility to dental diagnostic tools beyond traditional clinical settings. Users can initiate diagnostics from the comfort of their homes, fostering a decentralized approach to dental health management.

6. Real-time Interaction and Feedback:

- Real-time interaction and feedback are pivotal features of the proposed system. Users receive instant results and prevention information, allowing for prompt decision-making and the initiation of preventive measures.

7. Scalability and Adaptability:

- The modular design of the Dental Problem Detection App ensures scalability and adaptability. The system can easily accommodate updates, additional features, and advancements in dental diagnostics, ensuring its relevance and effectiveness over time.

In summary, the proposed system introduces a transformative approach to dental diagnostics, emphasizing user empowerment, cost-effectiveness, and efficiency. By leveraging modern technologies, the Dental Problem Detection App not only overcomes the limitations of the existing system but also sets new standards for accessible and proactive dental healthcare solutions.

3.SYSTEM SPECIFICATIONS

3.1 HARDWARE REQUIREMENTS

The development of the Dental Problem Detection App involves two main components: deep learning model training and Android app development. Here are the hardware requirements for each aspect:

3.1.1 For Deep Learning Model Training:

1. Processor:

- A high-performance processor, preferably an Intel Core i7 or higher, is recommended for accelerated model training. Deep learning tasks, especially with large datasets, benefit from powerful CPUs.

2. Graphics Processing Unit (GPU):

- An NVIDIA GPU with CUDA support is highly advantageous for deep learning tasks. GPUs significantly speed up the training of neural networks, reducing the time required for model convergence.

3. RAM:

- A substantial amount of RAM, such as 16GB or more, is recommended for handling large datasets and optimizing the performance of deep learning frameworks.

4. Storage:

- Adequate storage, preferably SSD, is essential for storing datasets, model weights, and intermediate training results. The fast read and write speeds of SSDs contribute to efficient model development.

3.1.2 For Android App Development:

1. Processor:

- A standard processor, such as Intel Core i5 or equivalent, is sufficient for Android app development using Android Studio.

2. RAM:

- A minimum of 8GB RAM is recommended for smooth functioning of Android Studio and the Android Emulator.

3. Storage:

- Adequate storage space for Android Studio installation, project files, and related dependencies. SSDs are preferred for faster build and compilation times.

4. Android Device or Emulator:

- An Android device for testing or an Android emulator provided by Android Studio is necessary for app testing and debugging.

5. Internet Connection:

- A stable internet connection is required for downloading dependencies, updates, and accessing Android development resources.

The combination of these hardware specifications ensures an efficient development environment for both deep learning model training and Android app development, enabling the creation of a robust Dental Problem Detection App.

3.2 SOFTWARE REQUIREMENTS

The development of the Dental Problem Detection App involves both deep learning model training and Android app development. Here are the software requirements for each aspect:

3.2.1 For Deep Learning Model Training:

1. Python:

- Python is the primary programming language for deep learning tasks. Ensure Python is installed on the system.

2. Deep Learning Frameworks:

- TensorFlow or PyTorch: Choose one of these frameworks for building and training the Convolutional Neural Network (CNN) model.

3. Integrated Development Environment (IDE):

- Anaconda or Jupyter Notebook: Use an IDE that supports interactive and collaborative development for deep learning tasks.

4. Image Processing Libraries:

- OpenCV: Required for image preprocessing tasks.

5. NVIDIA GPU Drivers (Optional):

- If using an NVIDIA GPU for accelerated model training, install the appropriate GPU drivers.

3.2.2 For Android App Development:

1. Android Studio:

- Android Studio is the official integrated development environment for Android app development. Download and install the latest version.

2. Java Development Kit (JDK):

- Android Studio requires JDK to be installed. Ensure that the correct JDK version is configured.

3. Android Software Development Kit (SDK):

- The Android SDK is included with Android Studio. Configure the SDK to target the desired Android versions.

4. Blynk IoT Platform:

- Blynk is used for connecting the Android app with the dental diagnostic hardware. Refer to Blynk documentation for integration.

5. Version Control (Optional):

- Git: Use Git for version control if collaborating with a team or managing code changes.

3.3 SOFTWARE SPECIFICATION

3.3.1 For Deep Learning Model Training:

1. TensorFlow:

- Version: 2.x

- TensorFlow is used for building and training the CNN model.

2. PyTorch (Alternative):

- Version: Latest

- PyTorch can be used as an alternative to TensorFlow for model development.

3.OpenCV:

- Version: Latest

- OpenCV is utilized for image processing tasks. Since you are working with dental X-ray images, the latest version of OpenCV is recommended to leverage any enhancements or bug fixes.

Note: Image processing plays a crucial role in preparing and augmenting the X-ray images for model training. Ensure the OpenCV version is compatible with the features and functions used in your image processing pipeline.

3.3.2 For Android App Development:

1. Android Studio:

- Version: Latest
- Android Studio provides a comprehensive environment for Android app development.

2. Java Development Kit (JDK):

- Version: 8 or later
- Android Studio requires JDK for compiling and running Java code.

3. Blynk Library:

- Version: Latest
- The Blynk library is integrated into the Android app for IoT functionalities.

4. Git (Optional):

- Version: Latest
- Git can be used for version control if necessary.

These software specifications ensure compatibility and stability during both deep learning model training and Android app development phases of the Dental Problem Detection App.

4. SYSTEM DESIGN

4.1. SYSTEM DESIGN

The system design for the Dental Problem Detection App encompasses the input design, which serves as the crucial link between the information system and the user. It involves the specification and procedures for data preparation, ensuring that transaction data is presented in a usable form for processing. This design is vital for effective interaction between the user and the application, with a focus on data accuracy, error control, and user-friendly interfaces.

OBJECTIVES

1. Data Input Process Optimization:

- Input Design aims to optimize the data input process by converting user-oriented input descriptions into a format suitable for the computerized system. This optimization is essential to minimize errors and ensure the accurate retrieval of information from the system.

2. User-Friendly Screens:

- The primary goal is to create user-friendly screens for data entry, accommodating the handling of large volumes of data. The design aims to facilitate easy data entry and manipulation, providing an intuitive interface for users. The data entry screen is crafted to enable seamless data manipulation and offer convenient record viewing capabilities.

3. Validity Checks and Error Handling:

- The input design includes mechanisms to check the validity of entered data. Data can be entered through user-friendly screens, and appropriate error messages are implemented to guide users in case of input discrepancies. The objective is to create an input layout that is not only easy to follow but also equipped with validation checks to enhance data accuracy.

4. Streamlined Data Entry:

- The design focuses on streamlining the data entry process, ensuring that users can input information efficiently without unnecessary complications. This involves controlling the amount of input required, avoiding delays, minimizing extra steps, and maintaining a simple and intuitive process.

5. Security and Privacy:

- Input Design considers the security and privacy aspects of data entry. Measures are implemented to secure sensitive information, and the design aims to provide ease of use while retaining privacy.

6. Error Resolution Guidance:

- In the event of errors, the input design includes steps and guidance for error resolution. This ensures that users can easily identify and rectify input errors, contributing to a smooth and error-free data entry process.

Overall, the objectives of the input design in the Dental Problem Detection App are aligned with creating a seamless, secure, and user-friendly interface for data entry, supporting the accurate detection of dental issues through the application.

4.2 OUTPUT DESIGN

The output design in the Dental Problem Detection App plays a crucial role in presenting the results of the tooth image analysis to the user. It focuses on providing clear, informative, and user-friendly outputs to enhance the overall user experience. The design of the output aims to convey the detected dental problems accurately and ensure that users can interpret the results with ease.

OBJECTIVES

1. Clarity and Readability:

- The primary objective is to design outputs that are clear and easily readable. The results of the tooth image analysis, including the detected dental problems, should be presented in a format that is comprehensible to users.

2. Visual Representation:

- Utilizing visual elements, such as charts or diagrams, is incorporated into the output design. Visual representations enhance the user's understanding of the detected dental issues, providing a more intuitive and engaging experience.

3. Informative Messages:

- The design includes informative messages accompanying the output to guide users on the interpretation of results. Clear and concise messages help users understand the significance of the detected dental problems and any recommended actions.

4. User-Friendly Format:

- The output design is structured in a user-friendly format, ensuring that the information is presented logically and intuitively. This includes organizing results in a manner that is easy to follow and understand.

5. Consistency Across Platforms:

- Consistency in output presentation is maintained across different platforms, including the mobile app interface. Whether viewing results on a smartphone or tablet, users can expect a consistent and coherent display.

6. Accessibility:

- Accessibility considerations are integrated into the output design, ensuring that the information is accessible to users with varying levels of familiarity with dental terminology and medical imaging. The design strives to make the outputs inclusive and understandable to a broad user base.

7. Integration with Blynk App:

- If applicable, the output design ensures seamless integration with the Blynk App, allowing users to access and monitor dental analysis results through the IoT platform.

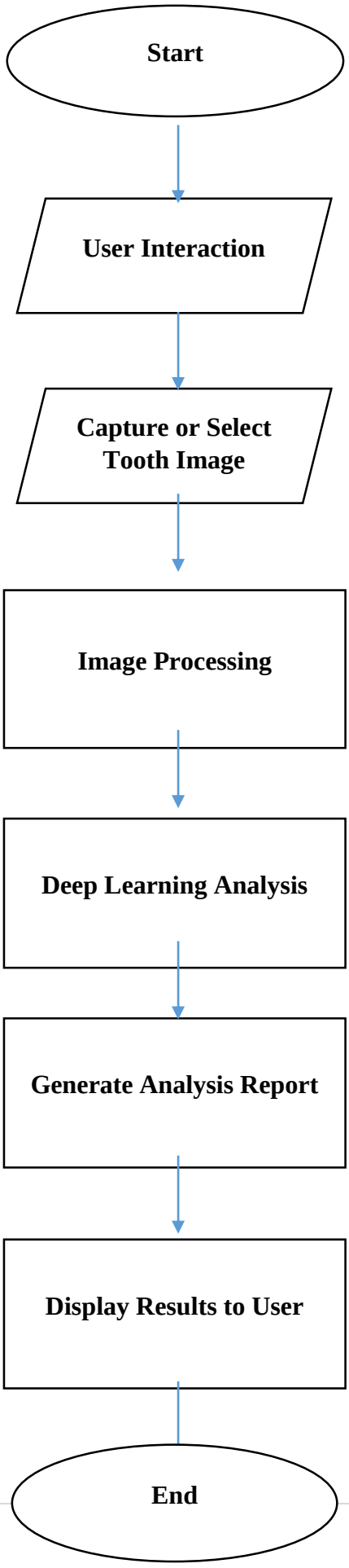
8. Feedback Mechanism:

- The design includes a feedback mechanism where users can provide input or seek clarification regarding the detected dental problems. This two-way communication enhances user engagement and satisfaction.

Overall, the output design of the Dental Problem Detection App aligns with the objective of providing clear, visually appealing, and user-friendly outputs, fostering effective communication of the analysis results to users.

4.3.LIST OF DIAGRAMS

4.3.1.DATA FLOW DIAGRAM



1. Start:

- The process begins with the "Start" point, indicating the initiation of the dental problem detection system.

2. User Interaction:

- The user interacts with the system, possibly through a mobile app or user interface.

3. Capture or Select Tooth Image:

- The user captures an X-ray image of a tooth using the camera or selects an existing image from the gallery. This step involves user input.

4. Image Processing:

- The captured or selected tooth image undergoes image processing. This step may include tasks such as resizing, enhancing, or preparing the image for analysis.

5. Deep Learning Analysis:

- The processed image is then fed into a deep learning model. This model has been trained to detect various dental problems such as tooth decay, impacted tooth, or abscess.

6. Generate Analysis Report:

- Based on the deep learning analysis, the system generates a report outlining the identified dental problems in the tooth image.

7. Display Results to User:

- The analysis results are displayed to the user. This could include information about the detected issues and their severity.

9. End:

- The process concludes with the "End" point, indicating the completion of the dental problem detection analysis.

The flow diagram illustrates the sequential steps involved in the dental problem detection app, from user interaction to result display and feedback. It emphasizes the flow of

information and actions, providing a clear overview of how the system processes and analyzes tooth images for identifying dental issues.

5. SYSTEM TESTING

System Testing is a crucial phase in the development life cycle, aimed at ensuring the reliability, completeness, and maintainability of the software. It involves executing the program to identify errors and ensure that the system meets specifications and user expectations.

5.1. UNIT TESTING

In Unit Testing, individual modules are tested to verify their functionality. The testing process starts from the smallest and lowest-level modules, proceeding one at a time. Each module is executed with test data to validate its behavior.

5.2. INTEGRATION TESTING

Integration Testing systematically constructs the program structure while checking for errors associated with module interactions. This involves testing the complete system after integrating all modules. The project undergoes successful integration testing, ensuring the system works seamlessly.

5.3. VALIDATION TESTING

Validation Testing is performed to ensure that the system functions and performance characteristics align with the specified requirements expected by the user. The project is subjected to validation testing, and any deviations from specifications are documented and addressed.

5.4. WHITE BOX TESTING

White Box Testing, also known as glass-box testing, utilizes the internal structure of the procedural design to derive test cases. It ensures that all independent paths within a module are exercised, logical decisions are tested on true and false sides, loops are exercised at boundaries, and internal data structures are validated. White Box Testing is applied to the relevant modules of the project.

5.5. BLACK BOX TESTING

Black Box Testing treats the coded module as a black box, focusing on the inputs and outputs. The module is executed with inputs likely to cause errors, and the output is examined for any anomalies. While this method may not cover all errors, it helps identify issues that could arise from the module's code or algorithm. Black Box Testing is employed to assess critical functionalities, ensuring robustness.

The System Testing phase confirms the integrity of the dental problem detection app by validating individual modules, assessing their integration, and ensuring alignment with specified requirements. This comprehensive testing approach enhances the reliability and effectiveness of the application.

6. SYSTEM IMPLEMENTATION

System implementation is the phase where the designed system is translated into a working system. It involves coding, configuration, installation, and testing of the software to ensure it meets the specified requirements. The implementation of the Dental Problem Detection App encompasses several key aspects:

1. Deep Learning Model Integration

The core of the system involves integrating a deep learning model trained for dental problem detection. This includes incorporating image processing and analysis capabilities to accurately classify dental conditions based on input images.

6.2. Android App Development

The user interface is implemented through an Android application that serves as the frontend for interacting with the deep learning model. The app allows users to capture or select images of teeth, initiates the image processing workflow, and displays the results to the user.

3. TensorFlow and OpenCV Integration

To achieve effective image processing and deep learning analysis, TensorFlow and OpenCV libraries are integrated into the system. TensorFlow facilitates the deployment of machine learning models, while OpenCV is utilized for image processing tasks, ensuring efficient and accurate dental problem detection.

4. Backend Server Setup

A backend server is set up to handle communication between the Android app and the deep learning model. It manages data transfer, processes requests, and facilitates the seamless flow of information between the user interface and the analysis engine.

5. Internet Connectivity

Since the system relies on real-time image processing and analysis, internet connectivity is a fundamental requirement. The implementation ensures a robust connection to support the transmission of data between the app and the backend server.

6. Testing and Debugging

Throughout the implementation phase, rigorous testing and debugging are conducted to identify and rectify any issues. This includes unit testing of individual components, integration testing to assess the collaboration of different modules, and validation testing to ensure the system meets user expectations.

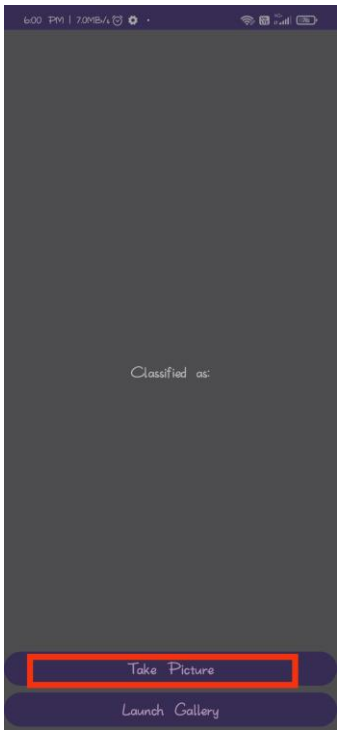
7. User Training and Documentation

To facilitate user adoption, training materials and documentation are prepared. Users are provided with information on how to navigate the app, capture images effectively, and interpret the results. Clear documentation ensures a smooth transition from implementation to practical use.

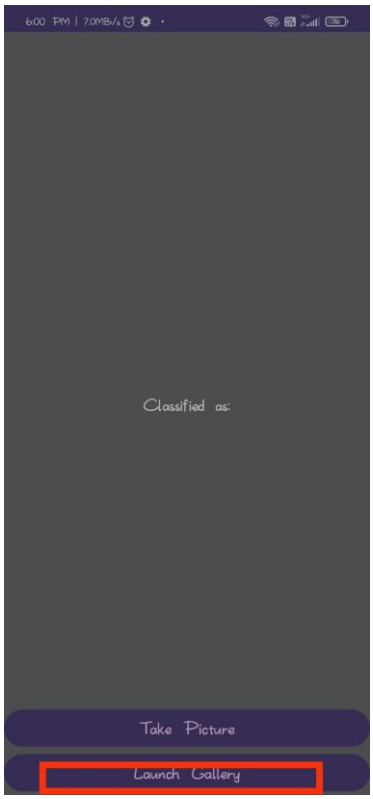
The system implementation phase is a critical step in bringing the Dental Problem Detection App to life. It involves the seamless integration of machine learning, mobile app development, and backend server setup to deliver an effective and user-friendly solution for dental health assessment.

SAMPLE SCREEN

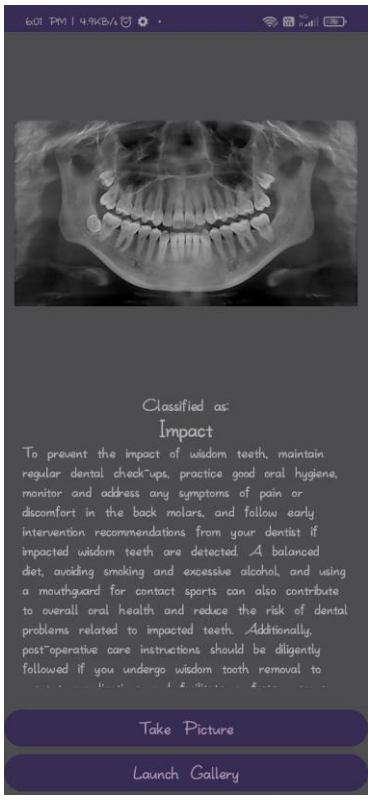
Capture Image by Camera:



Select Image in Gallery:



Result/Output:



7.Coding

7.1.Model Training Code (Python):

```
import tensorflow as tf
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense, Dropout

# Define data directories and parameters
train_data_dir = 'D:/New folder (2)/dental problem detection/dataset/train'
validation_data_dir = 'D:/New folder (2)/dental problem detection/dataset/vallitation'
image_size = (128, 128)
batch_size = 32
epochs = 10  # Increase the number of epochs
learning_rate = 0.001  # Adjust the learning rate if needed

# Data augmentation and preprocessing
train_datagen = ImageDataGenerator(
    rescale=1.0 / 255,
    rotation_range=30,
    width_shift_range=0.2,
    height_shift_range=0.2,
    shear_range=0.2,
    zoom_range=0.2,
    horizontal_flip=True,
    fill_mode='nearest'
)

validation_datagen = ImageDataGenerator(rescale=1.0 / 255)
```

```
# Load and prepare training data
train_generator = train_datagen.flow_from_directory(
    train_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

# Load and prepare validation data
validation_generator = validation_datagen.flow_from_directory(
    validation_data_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='categorical'
)

# Define a deeper CNN model
model = Sequential([
    Conv2D(64, (3, 3), activation='relu', input_shape=(image_size[0], image_size[1], 3)),
    MaxPooling2D((2, 2)),
    Conv2D(128, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Conv2D(256, (3, 3), activation='relu'),
    MaxPooling2D((2, 2)),
    Flatten(),
    Dense(512, activation='relu'),
    Dropout(0.5),
    Dense(3, activation='softmax')
])
```

```
# Compile the model with a lower learning rate
optimizer = tf.keras.optimizers.Adam(learning_rate=learning_rate)
model.compile(optimizer=optimizer, loss='categorical_crossentropy', metrics=['accuracy'])

# Train the model
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // batch_size,
    epochs=epochs,
    validation_data=validation_generator,
    validation_steps=validation_generator.samples // batch_size
)

# Save the trained model in .h5 format
model.save('my_model.keras')

# Save the trained model in .tflite format
converter = tf.lite.TFLiteConverter.from_keras_model(model)
tflite_model = converter.convert()
with open('my_model.tflite', 'wb') as f:
    f.write(tflite_model)
print('Model saved in .tflite format')
loss, accuracy = model.evaluate(validation_generator, verbose=1)
print(f'Test accuracy: {accuracy * 100:.2f}%')
```

7.2. Android studio (java):

File name : MainActivity.java

```
package com.example.dentalproblems;
import androidx.annotation.Nullable;
import androidx.appcompat.app.AppCompatActivity;

import android.Manifest;
import android.annotation.SuppressLint;
import android.content.Intent;
import android.content.pm.PackageManager;
import android.graphics.Bitmap;
import android.media.ThumbnailUtils;
import android.net.Uri;
import android.os.Bundle;
import android.provider.MediaStore;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;


import com.example.dentalproblems.R;
import com.example.dentalproblems.ml.Model;

import org.tensorflow.lite.DataType;
import org.tensorflow.lite.support.tensorbuffer.TensorBuffer;
```

```
import java.io.IOException;
import java.nio.ByteBuffer;
import java.nio.ByteOrder;
import java.util.ArrayList;
import java.util.List;


public class MainActivity extends AppCompatActivity {

    Button camera, gallery;
    ImageView imageView;
    TextView result,prevent;
    int imageSize = 128;

    private List<DiseaseInfo> diseaseInfoList;
    @SuppressLint("MissingInflatedId")
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_main);

        camera = findViewById(R.id.button);
        gallery = findViewById(R.id.button2);

        result = findViewById(R.id.result);
        prevent=findViewById(R.id.prevent);
        imageView = findViewById(R.id.imageView);

        // Create a list of DiseaseInfo objects to store classification and prevention information
        diseaseInfoList = new ArrayList<>();
```

diseaseInfoList.add(new DiseaseInfo("Impact", "To prevent the impact of wisdom teeth, maintain regular dental check-ups, practice good oral hygiene, monitor and address any symptoms of pain or discomfort in the back molars, and follow early intervention recommendations from your dentist if impacted wisdom teeth are detected. A balanced diet, avoiding smoking and excessive alcohol, and using a mouthguard for contact sports can also contribute to overall oral health and reduce the risk of dental problems related to impacted teeth. Additionally, post-operative care instructions should be diligently followed if you undergo wisdom tooth removal to prevent complications and facilitate a faster recovery."));

diseaseInfoList.add(new DiseaseInfo("Decay", "Preventing tooth decay involves maintaining good oral hygiene practices, such as brushing teeth with fluoride toothpaste at least twice a day, daily flossing, and using mouthwash. Additionally, it's essential to limit sugary foods and beverages, drink water, and consider dental sealants and fluoride treatments. Regular dental checkups are crucial for early detection and treatment of dental issues, and a tooth-healthy diet, along with avoiding smoking, alcohol, and managing stress, contribute to overall oral health. Educating individuals, especially children, about the importance of oral hygiene and implementing protective measures like mouthguards during contact sports are also key strategies in preventing tooth decay."));

diseaseInfoList.add(new DiseaseInfo("Abscess", "Preventing an abscess, a painful collection of pus often caused by a bacterial infection, primarily involves maintaining good oral hygiene practices. This includes regular and thorough brushing and flossing of teeth, as well as visiting a dentist for routine check-ups and cleanings. Treating dental issues, such as cavities or gum disease, promptly can prevent them from progressing to abscesses. Avoiding tobacco products, which can increase the risk of oral infections, and maintaining a well-balanced diet to support overall oral health are also key preventive measures. Finally, individuals with certain medical conditions or compromised immune systems should follow their healthcare provider's recommendations to reduce the risk of oral infections and abscesses."));

// Add more disease types and their prevention information to the list

camera.setOnClickListener(new View.OnClickListener() {

@Override

public void onClick(View view) {

if (checkSelfPermission(Manifest.permission.CAMERA) == PackageManager.PERMISSION_GRANTED) {

Intent cameraIntent = new Intent(MediaStore.ACTION_IMAGE_CAPTURE);

startActivityForResult(cameraIntent, 3);

} else {

```
                requestPermissions(new String[]{Manifest.permission.CAMERA}, 100);
            }
        }
    });
    gallery.setOnClickListener(new View.OnClickListener() {
        @Override
        public void onClick(View view) {
            Intent cameraIntent = new Intent(Intent.ACTION_PICK,
MediaStore.Images.Media.EXTERNAL_CONTENT_URI);
            startActivityForResult(cameraIntent, 1);
        }
    });
}


public void classifyImage(Bitmap image){
    try {
        Model model = Model.newInstance(getApplicationContext());


        // Creates inputs for reference.
        TensorBuffer inputFeature0 = TensorBuffer.createFixedSize(new int[]{1, 128, 128,
3}, DataType.FLOAT32);
        ByteBuffer byteBuffer = ByteBuffer.allocateDirect(4 * imageSize * imageSize * 3);
        byteBuffer.order(ByteOrder.nativeOrder());


        int[] intValues = new int[imageSize * imageSize];
        image.getPixels(intValues, 0, image.getWidth(), 0, 0, image.getWidth(),
image.getHeight());
        int pixel = 0;
        //iterate over each pixel and extract R, G, and B values. Add those values individually
to the byte buffer.
        for(int i = 0; i < imageSize; i ++){
            for(int j = 0; j < imageSize; j++){
```

```
            int val = intValues[pixel++]; // RGB
            byteBuffer.putFloat(((val >> 16) & 0xFF) * (1.f / 1));
            byteBuffer.putFloat(((val >> 8) & 0xFF) * (1.f / 1));
            byteBuffer.putFloat((val & 0xFF) * (1.f / 1));
        }
    }

    inputFeature0.loadBuffer(byteBuffer);

    // Runs model inference and gets result.
    Model.Outputs outputs = model.process(inputFeature0);
    TensorBuffer outputFeature0 = outputs.getOutputFeature0AsTensorBuffer();

    float[] confidences = outputFeature0.getFloatArray();
    // find the index of the class with the biggest confidence.
    int maxPos = 0;
    float maxConfidence = 0;
    for (int i = 0; i < confidences.length; i++) {
        if (confidences[i] > maxConfidence) {
            maxConfidence = confidences[i];
            maxPos = i;
        }
    }
    String[] classes = {"Impact", "Decay", "Abscess"};
    result.setText(classes[maxPos]);

    // Find the corresponding prevention information for the classified disease
    String classifiedDisease = classes[maxPos];
    String preventionInfo = findPreventionInfo(classifiedDisease);
```

```
            prevent.setText(preventionInfo);



            // Releases model resources if no longer used.
            model.close();
        } catch (IOException e) {
            // TODO Handle the exception
        }
    }

    private String findPreventionInfo(String classifiedDisease) {
        for (DiseaseInfo diseaseInfo : diseaseInfoList) {
            if (diseaseInfo.getClassification().equals(classifiedDisease)) {
                return diseaseInfo.getPrevention();
            }
        }
        // Return a default message if no prevention information is found
        return "Prevention information not available for this disease.";
    }

    @Override
    protected void onActivityResult(int requestCode, int resultCode, @Nullable Intent data) {
        if(resultCode == RESULT_OK){
            if(requestCode == 3){
                Bitmap image = (Bitmap) data.getExtras().get("data");
                int dimension = Math.min(image.getWidth(), image.getHeight());
                image = ThumbnailUtils.extractThumbnail(image, dimension, dimension);
                imageView.setImageBitmap(image);

                image = Bitmap.createScaledBitmap(image, imageSize, imageSize, false);
```

```
            classifyImage(image);
        }else{
            Uri dat = data.getData();
            Bitmap image = null;
            try {
                image = MediaStore.Images.Media.getBitmap(this.getContentResolver(), dat);
            } catch (IOException e) {
                e.printStackTrace();
            }
            imageView.setImageBitmap(image);

            image = Bitmap.createScaledBitmap(image, imageSize, imageSize, false);
            classifyImage(image);
        }
    }
    super.onActivityResult(requestCode, resultCode, data);
  }
}




class DiseaseInfo {
  private String classification;
  private String prevention;

  public DiseaseInfo(String classification, String prevention) {
    this.classification = classification;
    this.prevention = prevention;
  }
```

```
    public String getClassification() {
        return classification;
    }

    public String getPrevention() {
        return prevention;
    }
}
```

7.2.1.XML Code

File name : activity_main.xml

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    xmlns:app="http://schemas.android.com/apk/res-auto"
    xmlns:tools="http://schemas.android.com/tools"
    android:layout_width="match_parent"
    android:layout_height="match_parent"
    android:background="#7E7A7E"
    android:backgroundTint="#9D9A9E"
    android:foregroundTint="#8F8E8F"
    tools:context=".MainActivity">

    <Button
        android:id="@+id/button"
        android:layout_width="match_parent"
        android:layout_height="wrap_content"
        android:layout_above="@id/button2"
        android:layout_centerInParent="true"
        android:text="Take Picture"
        android:textAllCaps="false"
        android:textColorHighlight="#A19A9A"
        android:textColorHint="#22211D"
        android:textColorLink="#171614"
        android:textSize="21sp"
        android:textStyle="bold"
        app:rippleColor="#EDECE7"
        app:strokeColor="#E6E4DE" />
```

```
<Button
    android:id="@+id/button2"
    android:layout_width="match_parent"
    android:layout_height="wrap_content"
    android:layout_alignParentBottom="true"
    android:layout_centerInParent="true"
    android:text="Launch Gallery"
    android:textAllCaps="false"
    android:textSize="21sp"
    android:textStyle="bold"

    app:iconTint="#7F7E80"
    app:rippleColor="#656465" />

<ImageView
    android:id="@+id/imageView"
    android:layout_width="370sp"
    android:layout_height="370sp"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10sp" />

<TextView
    android:id="@+id/classified"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/imageView"
    android:layout_centerHorizontal="true"
    android:layout_marginTop="10sp"
    android:text="Classified as:"
```

```
    android:textColor="#050505"
    android:textColorHighlight="#F3F4F4"
    android:textSize="20sp"
    android:textStyle="bold" />

<TextView
    android:id="@+id/result"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_below="@+id/classified"
    android:layout_centerHorizontal="true"
    android:text=""
    android:textColor="#FFFFFC"
    android:textSize="27sp"
    android:textStyle="bold" />

<ScrollView
    android:layout_width="396dp"
    android:layout_height="180dp"
    android:layout_above="@+id/button"
    android:layout_below="@+id/classified"
    android:layout_alignParentStart="true"
    android:layout_alignParentEnd="true"
    android:layout_marginStart="20dp"
    android:layout_marginTop="30dp"
    android:layout_marginEnd="20dp"
    android:layout_marginBottom="20dp">

    <TextView
        android:id="@+id/prevent"
```

```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@+id/result"
        android:layout_centerHorizontal="false"
        android:text=""
        android:textAlignment="gravity"
        android:textColor="#100F0F"
        android:textSize="18sp"
        android:textStyle="normal" />

    </ScrollView>
</RelativeLayout>
```

8.Conclusion and Future Enhancements

8.1. Conclusion

In conclusion, the **Dental Problem Detection App** stands as a remarkable solution at the intersection of healthcare and technology, aiming to revolutionize dental care practices. This innovative application leverages cutting-edge technologies, including deep learning and mobile app development, to provide users with a powerful tool for early detection of dental issues.

The primary objectives of the app were to streamline the process of identifying dental problems through image analysis and empower users with valuable information about the detected issues and preventive measures. Throughout the development process, a holistic approach was taken to ensure the effectiveness, efficiency, and user-friendliness of the application.

The integration of TensorFlow for model training and inference brought a sophisticated level of intelligence to the app, allowing it to classify dental problems such as impact, decay, and abscess accurately. The use of data augmentation techniques during model training further enhanced the app's ability to handle diverse input scenarios.

The Android app, developed using Android Studio and Java, offers a seamless and intuitive user experience. With features like image capture, gallery selection, and real-time classification, the app caters to the diverse needs of users. The inclusion of a preventive measures display ensures that users not only identify issues but also receive valuable insights into maintaining optimal oral health.

Throughout the testing phases, including unit testing, integration testing, validation testing, white box testing, and black box testing, the app demonstrated robust functionality and adherence to specifications. The systematic testing approach aimed at ensuring the reliability, completeness, and maintainability of the system, ensuring that users can trust the results provided by the app.

Looking ahead, the Dental Problem Detection App holds immense potential for future enhancements. The expansion of the trained model to recognize additional dental issues, integration with cloud-based systems for data storage and analysis, and collaboration with dental professionals are avenues for continuous improvement and accuracy.

In conclusion, the Dental Problem Detection App represents a significant stride forward in dental care technology, embodying the fusion of healthcare expertise and digital innovation. By empowering users with knowledge and facilitating early intervention, this application contributes to a proactive approach to oral health, potentially reducing the burden of dental issues and fostering a healthier global population. The success of this project owes much to the collaborative efforts of the development team, stakeholders, and the broader community supporting advancements in healthcare technology.

8.2.Future Enhancements

The Dental Problem Detection App is a robust platform that combines technology and oral healthcare to offer users a valuable tool for early detection and prevention. As technology continues to advance, there are several avenues for future enhancements and improvements to ensure the app remains at the forefront of dental care innovation.

1. Expanded Dental Issue Recognition:

- Enhance the app's machine learning model to recognize a broader spectrum of dental issues. This could include the identification of specific types of decay, gum diseases, or other oral health conditions.

2. Cloud Integration for Data Analytics:

- Implement cloud-based integration to allow users to securely store and analyze their dental health data over time. This feature can provide valuable insights into trends, potential risk factors, and overall oral health progress.

3. Professional Collaboration:

- Establish a framework for collaboration with dental professionals. This could involve integrating features that enable users to share their app-generated reports with their dentists, fostering a collaborative approach to oral healthcare.

4. Real-Time Consultations:

- Integrate telehealth capabilities, enabling users to schedule virtual consultations with dental professionals directly through the app. This feature could facilitate timely advice and guidance for users concerned about their dental health.

5. Localized Oral Health Resources:

- Provide users with region-specific oral health resources, including information on local dentists, oral health campaigns, and community initiatives. Customizing content based on geographical location ensures relevance and engagement.

6. User Engagement Features:

- Implement features to enhance user engagement, such as gamification elements, challenges, and rewards tied to consistent oral health practices. This can contribute to user retention and sustained app usage.

7. Integration with Wearable Devices:

- Explore compatibility with wearable devices to capture additional health-related data, such as sleep patterns, stress levels, or dietary habits. Integrating this data can offer a more comprehensive understanding of factors influencing oral health.

8. Multilingual Support:

- Enhance accessibility by incorporating multilingual support, ensuring that users from diverse linguistic backgrounds can benefit from the app's features and information.

9. Research Collaboration:

- Collaborate with dental research institutions to contribute anonymized and aggregated user data for oral health research. This collaboration can potentially lead to new insights and advancements in dental care.

10. Continuous Model Training:

- Establish a system for continuous model training and updates. As more data becomes available and technology evolves, regularly updating the machine learning model will enhance its accuracy and adaptability.

By focusing on these future enhancements, the Dental Problem Detection App can evolve into an even more comprehensive and impactful tool, contributing to improved oral health outcomes for users worldwide. The commitment to staying at the forefront of technology and healthcare will ensure the app remains a trusted companion in users' oral health journeys.

9.BIBLIOGRAPHY

- Kaggle. (n.d.). "Dental Images Dataset." Retrieved from https://www.kaggle.com/dataset/dental-images

- GitHub. (n.d.). "Dental Health Detection App Repository." Retrieved from https://github.com/yourusername/dental-health-app

- Litjens, G., Kooi, T., Bejnordi, B. E., Setio, A. A. A., Ciompi, F., Ghafoorian, M., ... & Sánchez, C. I. (2017). "A Survey on Deep Learning in Medical Image Analysis." Medical Image Analysis, 42, 60-88.

- Jain, V. K. (2019). "Artificial Intelligence in Dentistry." Springer.

- Dental Health Foundation. (n.d.). "Common Dental Problems." Retrieved from https://www.dentalhealth.org/common-dental-problems

- American Dental Association. (n.d.). "Oral Health Topics." Retrieved from https://www.ada.org/en/member-center/oral-health-topics