Smoking Cancer Detection Mobile Application

By

Abdelrahman Yasser Mohamed Ibrahim	20-00184
Mohamed Ahmed Saeed Agamy	20-00312
Ahmed Hamada Hamdy Massoud	19-00764
Mohamed Mostafa Saad Ouf	20-00365
Mostafa Youssef Obaid Youssef	20-01130
Mohanad Nasser Abdelawal Ibrahim	18-00499
Diana Ashraf Wadea	20-00047

Supervised by

DR. Basem Mohamed El Omda

Eng. Mohamed Abdelmawgoud Khedr

TABLE OF CONTENTS

1.INTRODUCTION

1.1 Overview

Smoking and cancer have a deeply intertwined relationship, with smoking being one of the leading causes of cancer worldwide. Here's an overview:

1. **Causal Link**: Smoking is a well-established cause of several types of cancer. It contains thousands of chemicals, many of which are carcinogenic (cancer-causing). When these chemicals are inhaled, they can damage cells in the body, leading to the development of cancer over time.
2. **Types of Cancer**: Smoking is most strongly associated with lung cancer, accounting for the majority of cases. However, it also

significantly increases the risk of several other types of cancer, including:

- Bladder cancer
- Mouth and throat cancer
- Esophageal cancer
- Pancreatic cancer
- Stomach cancer
- Liver cancer
- Colorectal cancer
- Cervical cancer
- Kidney cancer

3. **Mechanism of Action**: The carcinogens in tobacco smoke can cause genetic mutations in the DNA of cells, disrupting normal cell function and leading to uncontrolled growth – the hallmark of cancer. Additionally, tobacco smoke can weaken the immune system's ability to destroy cancer cells, further promoting cancer development.
4. **Secondhand Smoke**: Even for non-smokers, exposure to secondhand smoke is linked to an increased risk of cancer, particularly lung cancer. Secondhand smoke contains many of the same cancer-causing chemicals found in directly inhaled smoke.
5. **Quitting Reduces Risk**: While the risk of cancer decreases significantly after quitting smoking, it may take years for the risk to drop to that of a non-smoker. However, the sooner one quits, the greater the reduction in cancer risk.

6. **Global Impact**: Smoking-related cancers represent a significant burden on public health systems worldwide. The World Health Organization (WHO) estimates that tobacco use is responsible for around 25% of cancer deaths globally.
7. **Prevention and Control**: Efforts to reduce smoking prevalence, such as tobacco taxation, advertising bans, smoke-free policies, and public awareness campaigns, are crucial in preventing smoking-related cancers. Early detection through screening programs and timely access to treatment also play vital roles in cancer control.

In summary, smoking is a major contributor to the global cancer burden, causing a wide range of cancers that lead to significant morbidity and mortality. Prevention efforts, including smoking cessation and public health interventions, are essential in reducing the incidence of smoking-related cancers and improving overall public health.

1.2 Problem

Detecting cancer in individuals who smoke presents several challenges:

1. **Late Diagnosis**: Smoking-related cancers often develop slowly and may not cause noticeable symptoms in the early stages. As a result, they are frequently diagnosed at later, more advanced stages when treatment options may be limited and prognosis poorer.
2. **Masking Symptoms**: Smoking can mask symptoms of certain cancers or mimic symptoms of other non-cancerous respiratory conditions, making it difficult to differentiate between smoking-related respiratory issues and early signs of cancer.

3. **High Prevalence of Other Conditions**: Smokers often suffer from other health conditions, such as chronic obstructive pulmonary disease (COPD) and cardiovascular disease, which can complicate the diagnostic process and delay cancer detection.
4. **Overlapping Risk Factors**: Smoking is associated with various lifestyle factors and environmental exposures that can also increase the risk of cancer. Distinguishing the contribution of smoking from other risk factors in cancer development can be challenging.
5. **Screening Limitations**: While screening methods such as chest X-rays, CT scans, and biomarker tests exist for some smoking-related cancers (e.g., lung cancer), they are not without limitations. False positives can lead to unnecessary invasive procedures and anxiety, while false negatives can result in missed opportunities for early intervention.
6. **Variability in Cancer Types and Presentation**: Smoking is linked to several types of cancer, each with its unique characteristics and presentation. Detecting and diagnosing these different types of cancers require a diverse set of screening and diagnostic approaches, adding complexity to the detection process.
7. **Stigma and Barriers to Healthcare**: Some individuals who smoke may face stigma or reluctance to seek medical care due to the association between smoking and certain cancers. This can result in delays in seeking medical attention and ultimately impact cancer detection and treatment outcomes.

Addressing these challenges requires a multi-faceted approach that includes public health efforts to promote smoking cessation, increased awareness of cancer symptoms among smokers and healthcare providers, improved access to screening and diagnostic tools, and ongoing research to develop better strategies for early detection and treatment of smoking-related cancers.

1.3 Previous Work

Research in smoking-related cancer detection has seen significant advancements over the years. Some notable areas of progress include:

Screening Technologies: Imaging techniques such as low-dose computed tomography (LDCT) have emerged as effective tools for early detection of lung cancer in high-risk individuals, including smokers. LDCT screening has been shown to reduce lung cancer mortality in this population.

Biomarker Discovery: Researchers have identified biomarkers, such as circulating tumor DNA (ctDNA) and specific proteins, that can indicate the presence of lung cancer and other smoking-related cancers. These biomarkers hold promise for improving early detection and monitoring treatment response.

Artificial Intelligence (AI): AI and machine learning algorithms are being developed to analyze medical imaging data, such as chest X-rays and CT scans, for early signs of lung cancer. These algorithms can assist radiologists in identifying suspicious lesions and improving the accuracy of cancer detection.

Liquid Biopsies: Liquid biopsy techniques, which analyze tumor-derived material circulating in bodily fluids like blood, urine, and saliva, offer a minimally invasive approach to cancer detection and monitoring. Liquid biopsies show potential for detecting early-stage lung cancer and monitoring treatment response in smokers.

Integration of Risk Assessment Tools: Risk prediction models, incorporating factors such as smoking history, age, and other demographic and clinical variables, are being developed to identify individuals at high risk of developing smoking-related cancers. These models help guide screening recommendations and early intervention strategies.

Multimodal Approaches: Combining different screening modalities, such as imaging, biomarker analysis, and risk assessment tools, may improve the sensitivity and specificity of cancer detection in smokers. Multimodal approaches allow for a more comprehensive evaluation of individuals at risk of smoking-related cancers.

Population-Based Screening Programs: Efforts are underway to implement population-based screening programs for lung cancer, particularly in high-risk populations such as current and former smokers. These programs aim to increase early detection rates and reduce mortality from smoking-related cancers through systematic screening and follow-up.

Overall, ongoing research in smoking-related cancer detection is focused on improving the sensitivity, specificity, and accessibility of screening methods to enable early detection and intervention, ultimately leading to improved outcomes for individuals at risk of developing these cancers.

1.4 Solution

A mobile application used to diagnose the patient's condition by answering a set of questions and then inserting a picture of a sample to determine if he is infected or not and if he is infected determines the stage of the disease

1.5 Objectives

4. **Early Detection and Intervention:**

The primary objective here is to identify signs of potential cancer in smokers at the earliest possible stage. This involves implementing screening programs such as regular chest X-rays, CT scans, or sputum cytology tests to detect any abnormalities in the lungs or airways. Early detection allows for prompt intervention and treatment, which can significantly improve outcomes and increase the chances of successful recovery.

2. User Education and Awareness:

This objective aims to educate smokers about the risks of developing cancer due to tobacco use and to raise awareness about the importance of regular health check-ups for early detection. Educational campaigns can provide information about the link between smoking and cancer, as well as resources for quitting smoking or reducing tobacco consumption. By increasing awareness, individuals are more likely to seek preventive healthcare and undergo screening tests for early cancer detection.

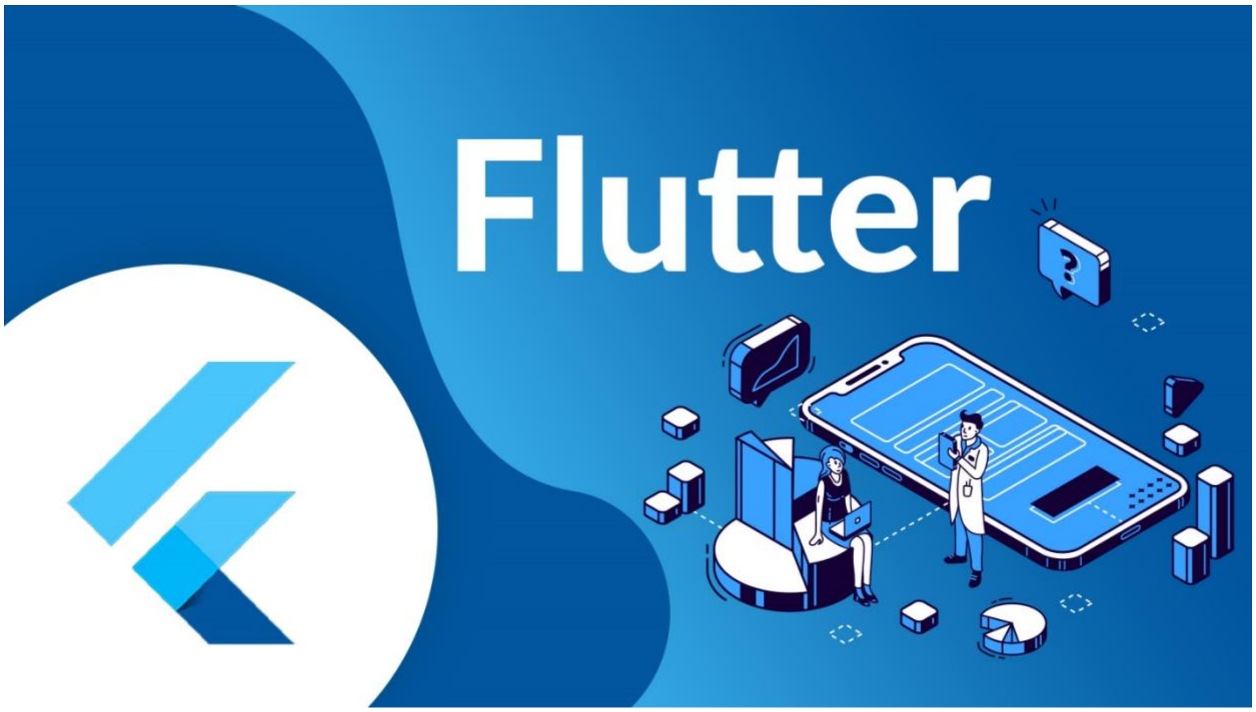
3. Behavioral Change Support:

Smoking cessation is the most effective way to reduce the risk of developing smoking-related cancers. This objective focuses on providing support and resources to help smokers quit or reduce their tobacco use. Behavioral change interventions, such as counseling, support groups, nicotine replacement therapy, or medication, can assist individuals in overcoming nicotine addiction and adopting healthier lifestyles. By supporting behavioral change, the aim is to reduce the incidence of smoking-related cancers and improve overall health outcomes.

4. Implement notifications and reminders for health check-ups and behavioral goals:

This objective involves using technology, such as mobile apps or automated messaging systems, to remind individuals about scheduled health check-ups, cancer screenings, and behavioral goals related to smoking cessation. Notifications can prompt smokers to attend appointments, undergo screening tests, or adhere to smoking cessation programs. By providing timely reminders and support, individuals are more likely to stay engaged in their healthcare and proactive in managing their risk factors for cancer.

CHAPTER TWO



2.1 Flutter

1- What is Flutter?

Flutter is a framework developed by Google to create web apps, mobile apps, and embedded apps using a single codebase. Flutter mobile apps perform well and can compete with native mobile apps. The first release of Flutter came in 2017, and the Flutter project was started under the name of Sky

2-What was the purpose of creating the Flutter framework?

The purpose of making the Flutter framework is to enable developers to create native-like performing mobile apps on multiple platforms with a single codebase.

The Flutter framework includes two important parts:

A Software Development Kit (Flutter SDK): The mobile SDK is a collection of software development tools that help developers to write a mobile app code. The tools available in Flutter are APIs for unit and integration tests, Dart DevTools, and Flutter & Dart command-line tools.

Flutter Widgets (A Library-based on Widgets): Widgets are a collection of reusable UI elements (text buttons, text inputs, sliders, scrolling, and styling). It allows developers to build an application's UI depending on the client's needs. Widgets help to design layout, handle user interaction, and UIViews of the app.

You can check a short brief on the history of the Flutter mobile app framework in the following image

3-Which programming language is used in the Flutter framework?

Flutter framework is built using a programming language called Dart.

3- How Does Flutter Work?

Flutter apps use the Dart language, and it allows to compile the app code ahead of time into a native machine code (iOS/Android). Then, the rendering engine (built with C++) converts app code into Android's NDK and iOS's LLVM. Both the pieces of code are rendered into the wrapper Android and iOS platforms and results into .apk or .ipa files for final output.

4- Why Should You Use Flutter for App Development?

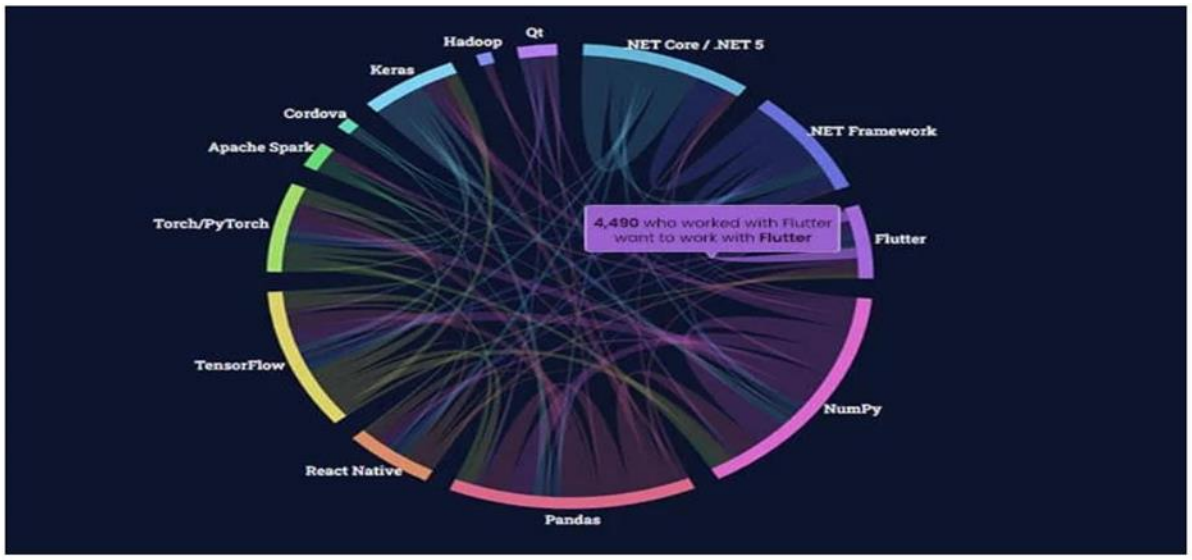
To give you reasons why you should use the Flutter framework, we have divided this section into two parts.

1. Some statistics on the Flutter framework

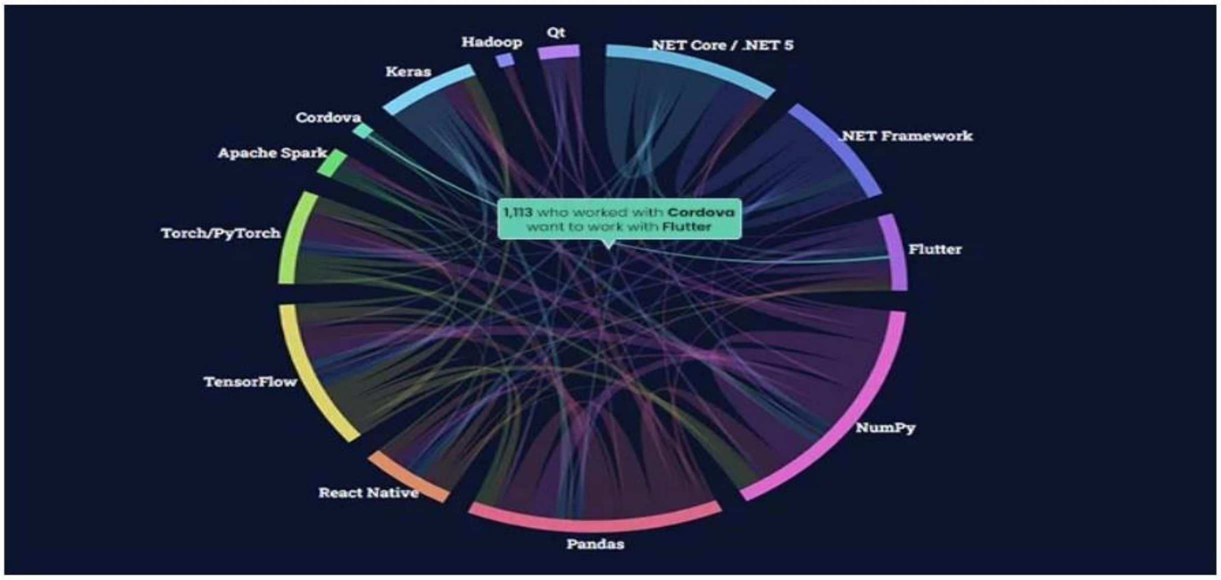
2. Features of Flutter framework

According to the StackOverflow survey 2021, 4,490 developers worked with Flutter.

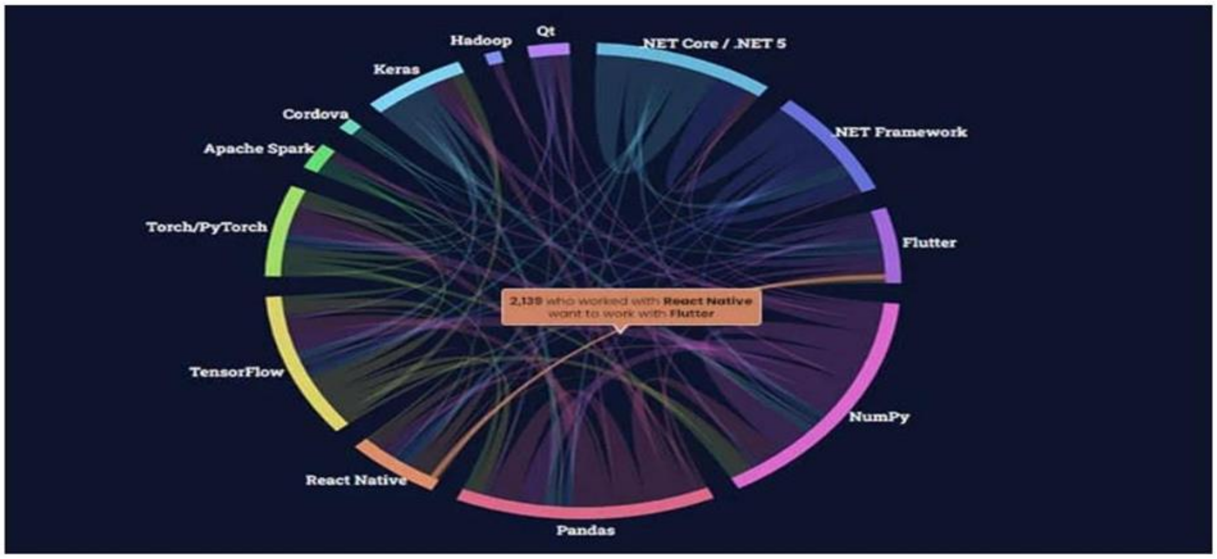
And all of these developers want to continue working with the Flutter framework.



Around 1,113 Cordova developers want to use the Flutter framework to develop cross-platform mobile apps



2,319 React Native developers want to use the Flutter framework for app development.



The above statistics describe that developers who are working with different frameworks or technologies also want to use the Flutter framework. Being one of the trending technologies to create native mobile apps, Flutter is used in more than 50,000 mobile apps, and these apps work on both platforms – Android and iOS.

5-Features of the Flutter app development framework

Flutter helps developers to develop fast and scalable apps. Following are the features.

- Flutter widgets
- AOT code compilation
- Hot reload
- Single codebase

6-What are the Advantages of Flutter for Android Developers?

1. Writing Code Faster
2. Single Codebase for Multiple Platforms
3. Architecture of Flutter
4. GUI and Widgets Components
5. Testing is Easy and Fast

FAQ About Flutter Framework

- Which are the top apps built using the Flutter framework?

- Google Ads
- Google Pay
- ByteDance
- eBay
- Nubank
- Toyota

1.Is it possible to create web applications or desktop apps using Flutter?

Yes, it is possible to create web applications and desktop apps using Flutter. The single code gets compiled and also works on the web.

2.Which is the best framework between Flutter and React Native for cross- platform mobile apps?

Both frameworks are best to build mobile apps. Flutter is famous because of its easy-to-use feature while React Native is famous for developing personalized UI of mobile apps.

3.Is Flutter better than Java?

Flutter is an SDK for creating cross-platform applications while Java only supports to development of Android apps.

4.Is Flutter easy to learn?

Flutter is easy to learn compared to Swift, Java programming language or React Native library. Because the learning curve of this framework is small compared to core language for Android or iOS app programming language.

Programming Language: Dart

Dart:

Dart is a programming language developed by Google, initially unveiled in 2011. It's primarily used for building web, server, and mobile applications. Dart was created with the goal of providing a productive, fast, and efficient alternative for developing modern applications.

Here are some key characteristics and features of Dart:

Object-Oriented:

Dart is an object-oriented language, meaning it supports concepts like classes, objects, inheritance, polymorphism, and encapsulation.

Strongly Typed:

Dart is a statically typed language, which means variables are explicitly declared with their data types. This helps catch errors at compile time and enhances code readability and maintainability.

Garbage Collected:

Dart uses automatic memory management through garbage collection, which helps developers focus on writing code without worrying about memory management issues like memory leaks.

Concurrency Support:

Dart provides built-in support for asynchronous programming, enabling developers to write concurrent code easily. This is crucial for handling tasks like network requests, file I/O, and UI updates without blocking the main thread.

JIT and AOT Compilation:

Dart supports both Just-In-Time (JIT) and Ahead-Of-Time (AOT) compilation. JIT compilation is used during development to enable features like hot reload, allowing developers to see changes instantly without restarting the application. AOT compilation, on the other hand, is used for production builds to generate highly optimized native machine code for better performance.

Cross-Platform Compatibility:

Dart is used as the primary language for developing applications using the Flutter framework. Flutter allows developers to build high-performance, cross-platform mobile applications for iOS, Android, and other platforms using a single codebase.

Tooling and Ecosystem:

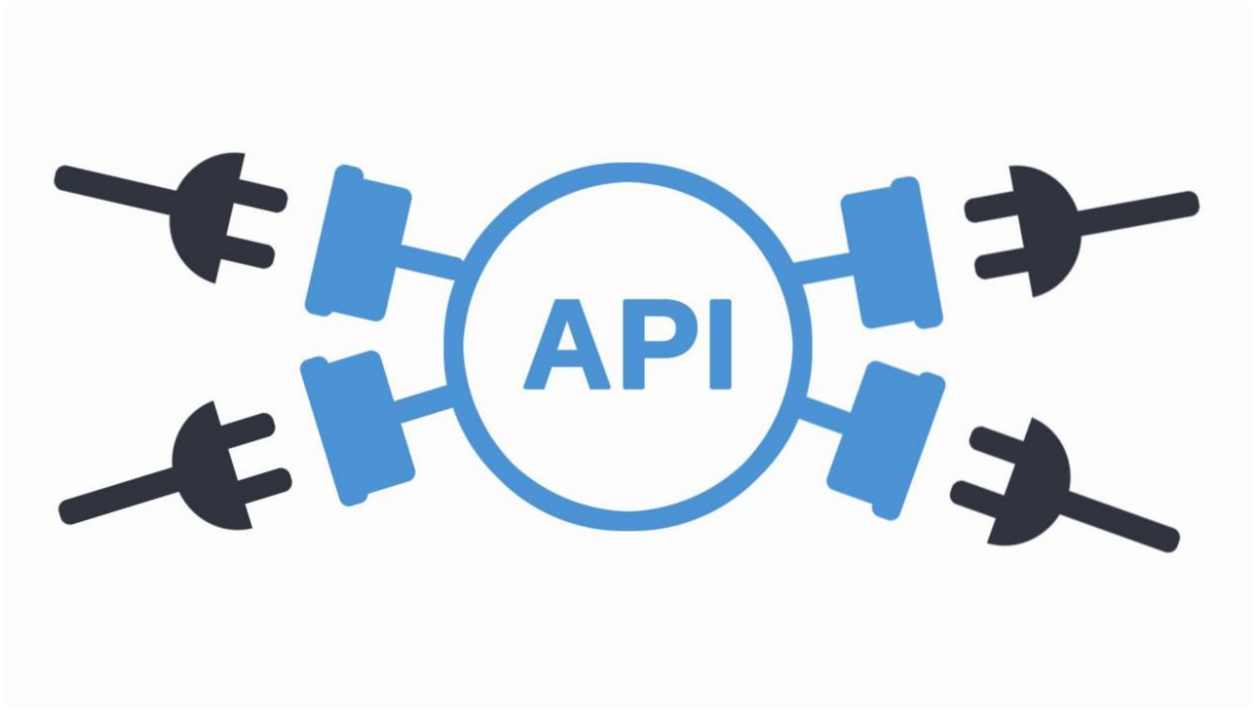
Dart comes with a rich set of development tools, including the Dart SDK, Dart VM (Virtual Machine), package manager (pub), and various IDE plugins for popular editors like Visual Studio Code and IntelliJ IDEA. Additionally, Dart has a growing ecosystem of packages and libraries available through Dart's package repository (pub.dev).

Overall, Dart aims to provide a productive and efficient development experience for building modern applications across different platforms, with a focus on performance, scalability, and maintainability.

Developers can deploy Dart apps in six ways:

Dart Deployment Methods

Deployment Type	**Target Platform**	**Platform specific**	**Requires Dart VM**	**Compile Speed**	**Execution Speed**
JavaScript	Browser	No	No	Slow	Fast
WebAssembly (preview)	Browser	No	No	Slow	Fast
Self-contained executable	macOS, Windows, Linux	Yes	No	Slow	Fast
Ahead-of-time module	macOS, Windows, Linux	Yes	No	Slow	Fast
Just-in-time module	macOS, Windows, Linux	Yes	Yes	Fast	Slow
Portable module	macOS, Windows, Linux	No	Yes	Fast	Slow



API

what is Flask ?

Flask is a micro web framework written in Python. It's lightweight and designed to be simple yet extensible, making it ideal for building web applications, including APIs. When Flask is used to build APIs, it allows developers to create endpoints that can handle HTTP requests and return data in various formats such as JSON.

Here are some key features of building APIs with Flask:

Routing:

Flask allows you to define routes for different endpoints of your API. You can specify the URL pattern for each endpoint and associate it with a function that handles the request and generates the response.

HTTP Methods:

Flask supports different HTTP methods like GET, POST, PUT, DELETE, etc. You can specify which methods are allowed for each endpoint, allowing you to create RESTful APIs.

Request Handling:

Flask provides a request object that contains all the information about the incoming HTTP request, such as headers, parameters, and body. You can access this object in your endpoint functions to process the request data.

Response Generation:

Similarly, Flask provides a response object that allows you to construct and customize the HTTP response that will be sent back to the client. This includes setting headers, status codes, and response body.

JSON Serialization:

Flask makes it easy to serialize Python objects to JSON format, which is commonly used for transmitting data in web APIs. You can use libraries like jsonify to convert dictionaries or lists to JSON and return them as part of the response.

Middleware:

Flask allows you to use middleware to intercept and process requests and responses before they reach your endpoint functions. This can be useful for implementing cross-cutting concerns like authentication, logging, or error handling.

Extension Ecosystem:

Flask has a rich ecosystem of extensions that add additional functionality to your API, such as authentication, database integration, serialization/deserialization, and more. These extensions can help streamline development and add features without reinventing the wheel.

Overall, Flask provides a flexible and efficient framework for building APIs in Python. Its simplicity, combined with its extensibility and robust ecosystem, makes it a popular choice for developers looking to create web services and APIs.

Algorithms

PyTorch
DYNAMIC NEURAL NETWORKS
HARDWARE ACCELERATED INFERENCE
EAGER & GRAPH-BASED EXECUTION
DISTRIBUTED TRAINING
SIMPLICITY OVER COMPLEXITY

PyTorch

What is PyTorch?

PyTorch is an open-source deep learning framework developed by Facebook's AI Research lab (FAIR). It's primarily used for building neural networks and conducting deep learning research, but it's also widely adopted in industry for various machine learning applications.

Here are some key features and characteristics of PyTorch:

Dynamic Computational Graphs:

One of the distinguishing features of PyTorch is its dynamic computation graph mechanism. Unlike some other deep learning frameworks that use static computation graphs (defined and compiled before execution), PyTorch builds computational graphs dynamically as operations are executed. This allows for more flexibility and ease in model development and debugging.

Tensor Computation:

PyTorch provides efficient tensor computation with support for GPU acceleration. Tensors are multi-dimensional arrays similar to NumPy arrays but optimized for deep learning tasks. PyTorch offers a rich set of operations for manipulating tensors and performing mathematical operations, making it suitable for a wide range of machine learning tasks.

Automatic Differentiation:

PyTorch's autograd package provides automatic differentiation, allowing gradients to be automatically computed for tensors with respect to some objective function. This is essential for training neural networks using gradient-based optimization algorithms like stochastic gradient descent (SGD) or variants such as Adam or RMSProp.

Neural Network Building Blocks:

PyTorch includes a rich library of neural network building blocks, such as layers, activation functions, loss functions, and optimization algorithms. These building blocks can be easily composed to create complex neural network architectures for various tasks, including classification, regression, image recognition, natural language processing, and more.

Dynamic Neural Networks:

With PyTorch, you can define and modify neural network architectures dynamically during runtime. This dynamic nature enables more experimentation and rapid prototyping, as models can be easily modified and adapted to different requirements without the need for recompilation.

Support for Deployment:

PyTorch provides tools and libraries for deploying trained models to production environments, including support for mobile devices (via PyTorch Mobile) and integration with deployment platforms such as TorchServe and ONNX (Open Neural Network Exchange).

Community and Ecosystem:

PyTorch has a vibrant and active community of researchers, developers, and enthusiasts. It offers extensive documentation, tutorials, and online resources to support users at all levels of expertise. Additionally, PyTorch has a rich ecosystem of third-party libraries and extensions for tasks like data loading, model interpretation, and visualization.

Overall, PyTorch is a powerful and flexible deep learning framework that offers both simplicity for beginners and advanced capabilities for researchers and practitioners in the field of artificial intelligence and machine learning. Its dynamic nature, intuitive API, and strong community support make it a popular choice for developing and deploying deep learning models.

Tensorflow

What is Tensorflow?

TensorFlow is an open-source machine learning framework developed by Google Brain team. It's one of the most popular and widely used frameworks for building and deploying machine learning models, particularly deep learning models.

Here are some key features and characteristics of TensorFlow:

Graph-Based Computation:

TensorFlow uses a static computational graph paradigm, where operations are defined as nodes in a graph, and data flows through the graph. This graph represents the computational dependencies between operations, allowing for efficient execution and optimization, especially for large-scale deep learning models.

Automatic Differentiation:

TensorFlow provides automatic differentiation through its tf.GradientTape API, enabling the computation of gradients for optimization algorithms such as gradient descent. This feature is essential for training neural networks using backpropagation.

Tensor Operations:

TensorFlow is built around the concept of tensors, which are multi-dimensional arrays similar to NumPy arrays. It provides a rich set of operations for tensor manipulation and mathematical computations, including element-wise operations, matrix operations, and reductions.

High-Level APIs:

TensorFlow offers high-level APIs like Keras, which provide a user-friendly interface for building and training neural networks. Keras allows developers to quickly prototype models using pre-built layers, activation functions, loss functions, and optimizers, while still offering flexibility for customization.

Scalability:

TensorFlow is designed for scalability and can run on a variety of hardware platforms, including CPUs, GPUs, and TPUs (Tensor Processing Units). It supports distributed computing, allowing models to be trained and deployed across multiple devices or machines for improved performance and efficiency.

Model Deployment:

TensorFlow provides tools and libraries for deploying trained models to production environments. This includes TensorFlow Serving for serving models via REST APIs, TensorFlow Lite for deploying models on mobile and embedded devices, and TensorFlow.js for running models in web browsers.

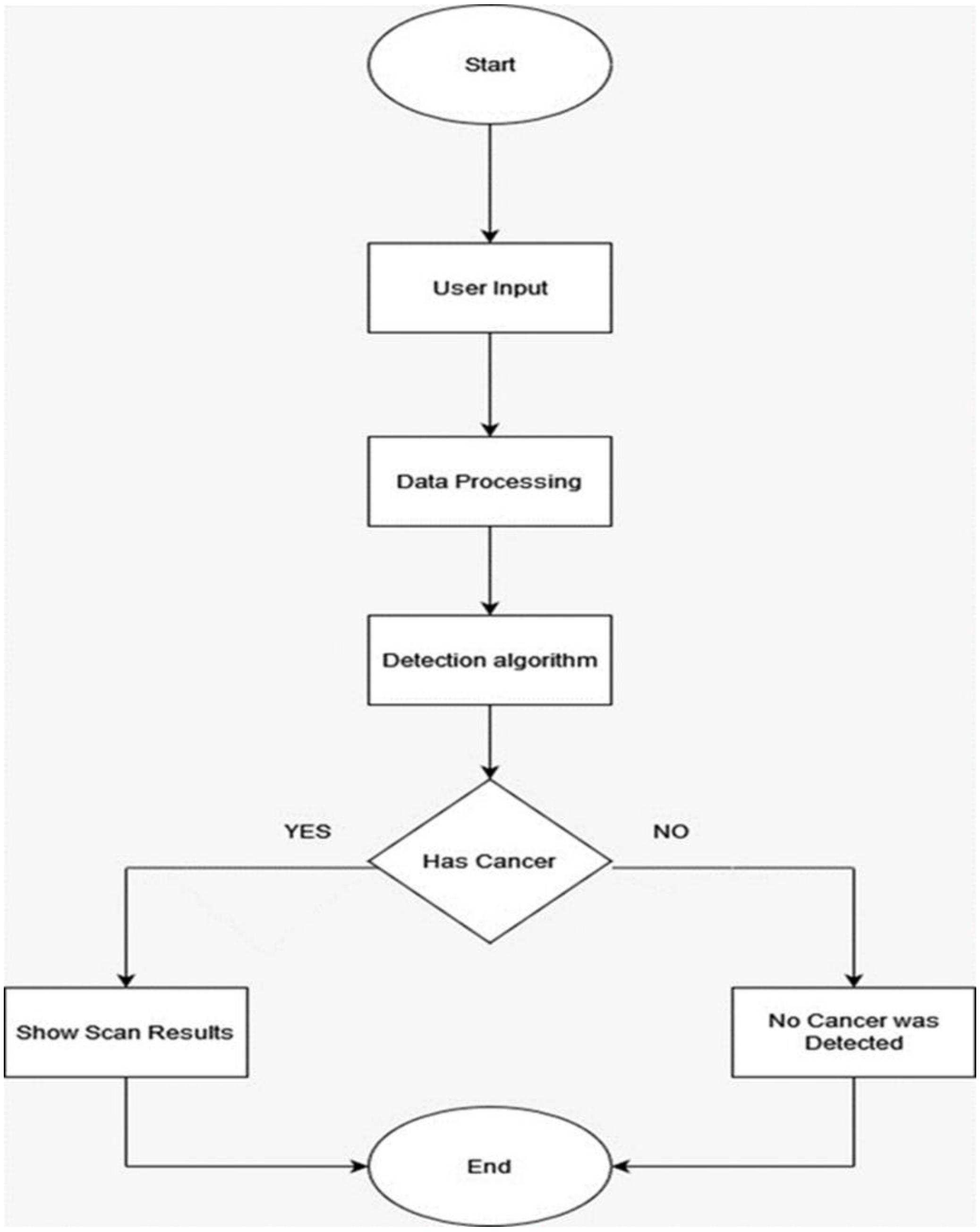
Community and Ecosystem:

TensorFlow has a large and active community of developers, researchers, and enthusiasts. It offers extensive documentation, tutorials, and online resources to support users at all levels of expertise. Additionally, TensorFlow has a rich ecosystem of third-party libraries and extensions for tasks like data preprocessing, model interpretation, and visualization.

Overall, TensorFlow is a powerful and versatile framework for building and deploying machine learning models, particularly deep learning models. Its scalability, flexibility, and extensive ecosystem make it a popular choice for a wide range of applications in research and industry.

CHAPTER THREE

Flow Chart



This flowchart represents a process for detecting cancer using an automated system. Here’s a detailed explanation of each step:

1. **Start:**

The process begins.

2. **User Input:**

The system receives input from the user. This could be in the form of medical data, images, or other relevant information necessary for cancer detection.

3. Data Processing:

The input data is processed to prepare it for analysis. This may involve cleaning the data, normalizing it, or performing other preprocessing steps to ensure it is in a suitable format for the detection algorithm.

4. Detection Algorithm:

The processed data is fed into a detection algorithm designed to identify cancer. This algorithm likely uses machine learning or other advanced computational techniques to analyze the data.

5. **Decision Point (Has Cancer):**

The algorithm makes a determination on whether the data indicates the presence of cancer. This is a decision point where the process branches into two possible outcomes:

- Yes (Has Cancer):

If the algorithm detects cancer, the process moves to the next step where the scan results are shown.

- Show Scan Results: The system displays the detailed scan results to the user, which includes the findings indicating the presence of cancer.

- No (No Cancer Detected):

If the algorithm does not detect cancer, the process proceeds to the step where the user is informed that no cancer was detected.

- No Cancer Was Detected:

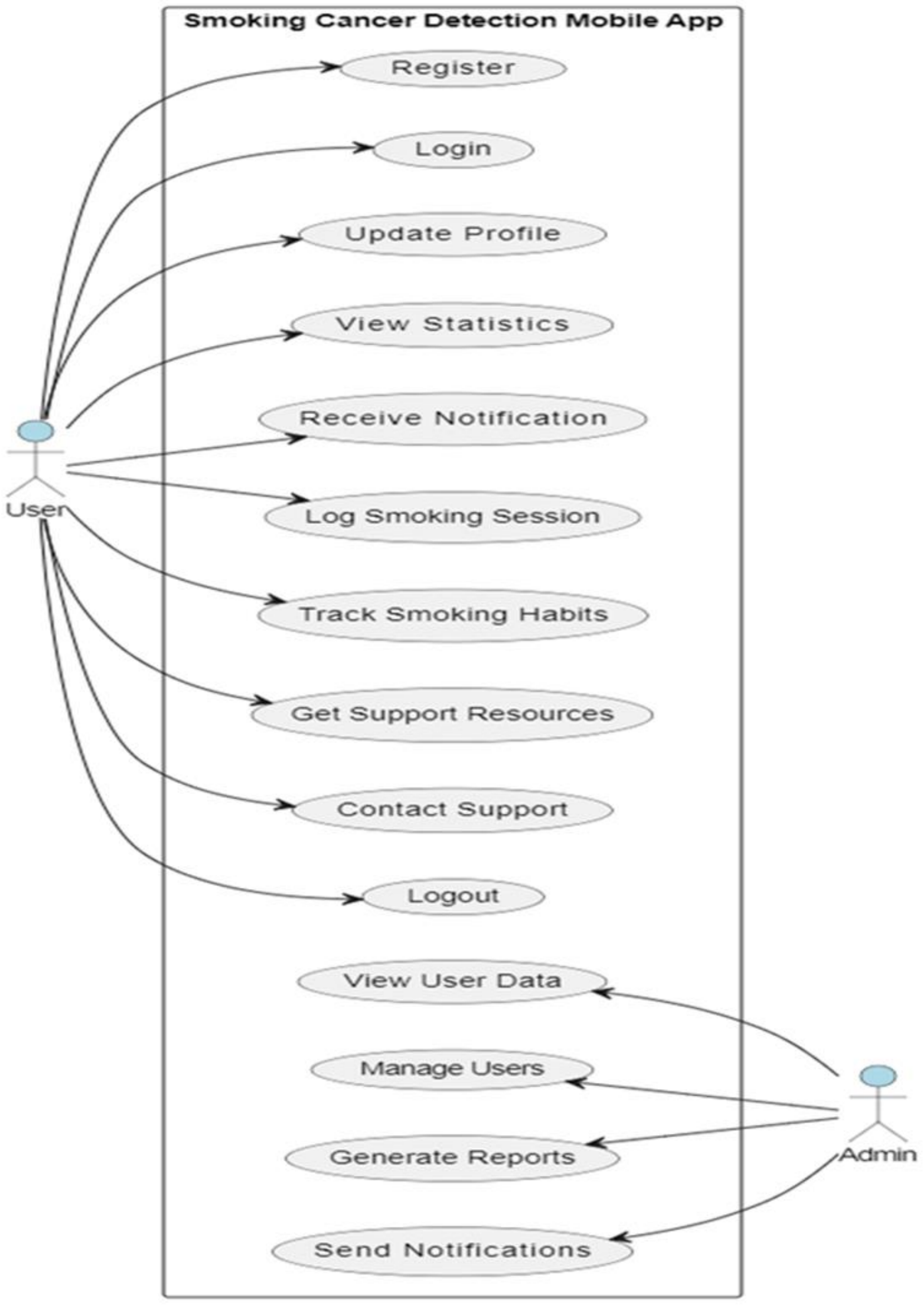
The system informs the user that no cancer was found in the data provided.

6. End:

The process concludes after the results are displayed to the user, whether cancer was detected or not.

In summary, this flowchart outlines a straightforward automated cancer detection process, starting from user input and data processing, through to the application of a detection algorithm, and finally presenting the results based on whether cancer was detected.

Use Case Diagram



This use case diagram depicts the interactions between users and administrators within a "Smoking Cancer Detection Mobile App." It highlights various functionalities available to both user roles. Here’s a detailed explanation of the diagram:

Actors

- User:

Represents a general user of the mobile app.

- Admin:

Represents an administrator who manages the app.

Use Cases for Users

1. Register:

Allows a new user to create an account on the app.

2. Login:

Enables users to sign in to their accounts.

3. Update Profile:

Users can update their personal information.

4. View Statistics:

Users can view their smoking statistics and progress over time.

5. Receive Notification:

Users receive notifications related to their smoking habits or health.

6. Log Smoking Session:

Users can log their smoking sessions, recording the time and amount smoked.

7. **Track Smoking Habits:**

Users can monitor and track their smoking patterns and habits.

8. **Get Support Resources:**

Provides users with resources and support for smoking cessation and cancer prevention.

9. **Contact Support:**

Allows users to contact support for assistance.

10. **Logout:**

Users can log out of their account.

Use Cases for Admins

1. **View User Data:**
   Admins can view data related to users, likely for monitoring and support purposes.
2. **Manage Users:**
   Admins can manage user accounts, including creating, updating, or deleting accounts.
3. **Generate Reports:**
   Admins can generate reports based on user data, which might be used for analysis and improving app services.
4. **Send Notifications:**
   Admins can send notifications to users, possibly for important updates or alerts.

Relationships

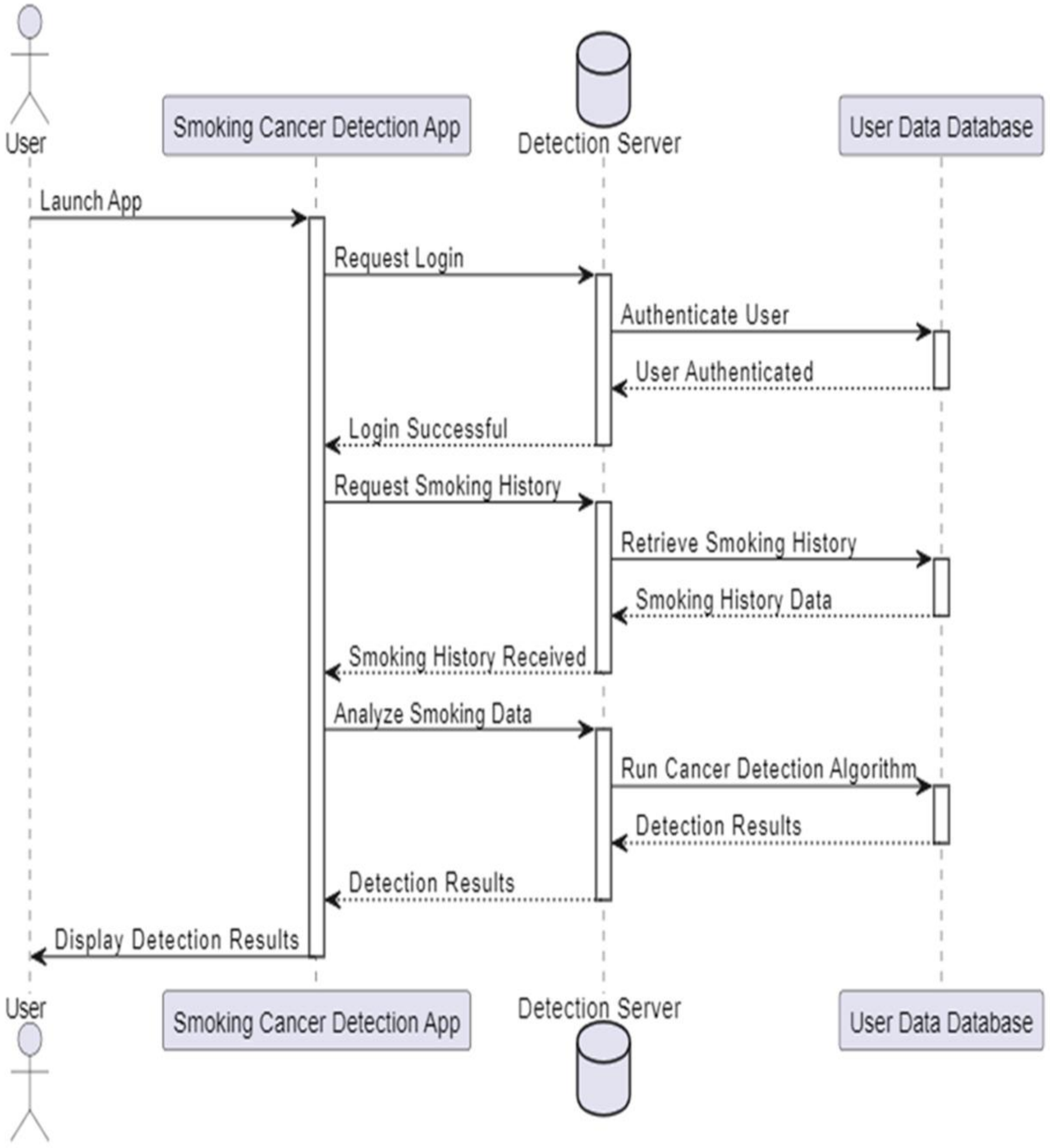
- Users and Admins are connected to their respective use cases, indicating they can perform these actions.

- There are lines connecting each actor to their corresponding use cases, showing the actions they are involved in.

Summary

The diagram illustrates the functionalities offered by the "Smoking Cancer Detection Mobile App," showing how users can interact with the app for logging and tracking smoking habits, viewing statistics, and getting support. It also shows the administrative capabilities for managing users, generating reports, and sending notifications. The overall purpose of the app appears to be aiding users in monitoring their smoking behavior and potentially detecting risks related to smoking and cancer.

Sequence Diagram



This sequence diagram illustrates the interactions between a user, the Smoking Cancer Detection App, the Detection Server, and the User

Data Database. The diagram details the process from launching the app to displaying cancer detection results. Here is a step-by-step explanation:

Participants

- User:

The individual using the app.

- Smoking Cancer Detection App:

The mobile application used by the user.

- Detection Server:

The server responsible for authenticating users, retrieving data, and running the detection algorithm.

- User Data Database:

The database storing user information and smoking history.

Process Flow

1. Launch App (User → Smoking Cancer Detection App)

   - The user launches the Smoking Cancer Detection App.

2. Request Login (Smoking Cancer Detection App → Detection Server)

   - The app sends a login request to the Detection Server to authenticate the user.

3. Authenticate User (Detection Server → User Data Database)

   - The Detection Server queries the User Data Database to authenticate the user's credentials.

   - User Authenticated (User Data Database → Detection Server)

     - The User Data Database confirms the user's authentication and sends the confirmation back to the Detection Server.

4. Login Successful (Detection Server → Smoking Cancer Detection App)

   - The Detection Server informs the app that the user has been successfully authenticated.

5. Request Smoking History (Smoking Cancer Detection App → Detection Server)

   - The app requests the user's smoking history from the Detection Server.

6. Retrieve Smoking History (Detection Server → User Data Database)

   - The Detection Server queries the User Data Database for the user's smoking history.

   - Smoking History Data (User Data Database → Detection Server)

     - The User Data Database returns the smoking history data to the Detection Server.

7. Smoking History Received (Detection Server → Smoking Cancer Detection App)

- The Detection Server sends the smoking history data to the app.

8. Analyze Smoking Data (Smoking Cancer Detection App)

- The app analyzes the received smoking data to prepare for running the cancer detection algorithm.

9. Run Cancer Detection Algorithm (Smoking Cancer Detection App → Detection Server)

- The app sends the analyzed smoking data to the Detection Server to run the cancer detection algorithm.

10. Detection Results (Detection Server → User Data Database)

- The Detection Server runs the cancer detection algorithm and stores the results in the User Data Database.

- Detection Results (User Data Database → Detection Server)

  - The User Data Database sends the detection results back to the Detection Server.

11. Detection Results (Detection Server → Smoking Cancer Detection App)

- The Detection Server sends the final detection results back to the app.

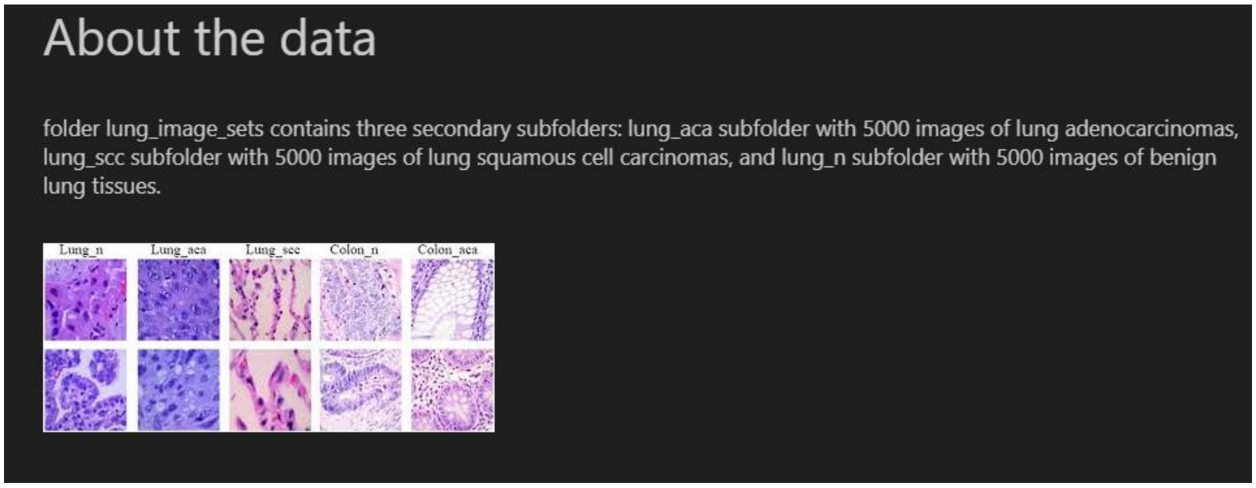
12. Display Detection Results (Smoking Cancer Detection App → User)

- The app displays the detection results to the user.

Summary

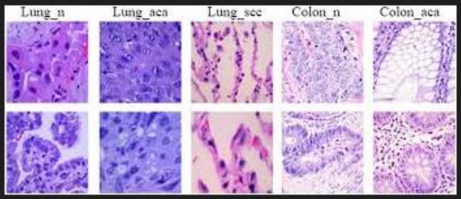
This sequence diagram shows a typical workflow for a smoking cancer detection mobile app, starting from user authentication and data retrieval to running a detection algorithm and displaying the results. Each step involves interactions between the user, app, server, and database to ensure accurate data processing and result presentation.

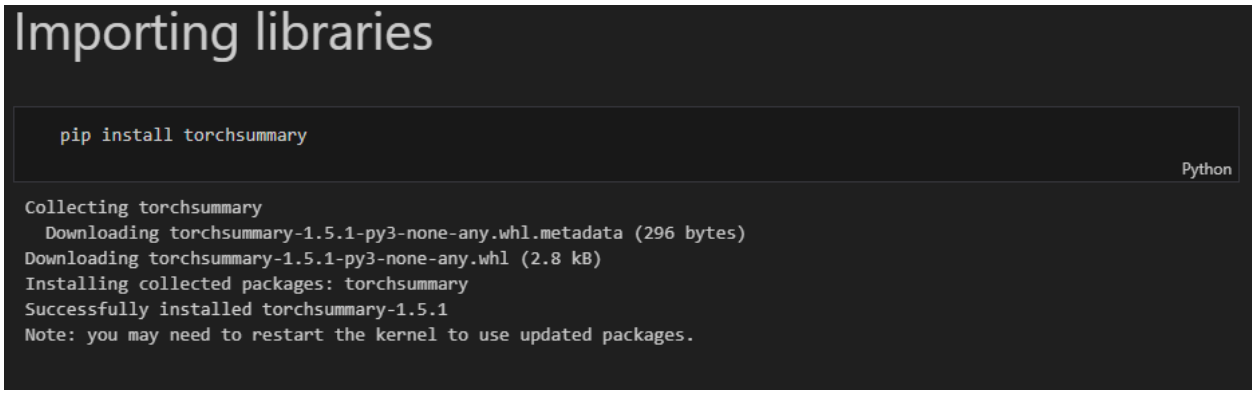
CHAPTER FOUR



About the data

folder lung_image_sets contains three secondary subfolders: lung_aca subfolder with 5000 images of lung adenocarcinomas, lung_scc subfolder with 5000 images of lung squamous cell carcinomas, and lung_n subfolder with 5000 images of benign lung tissues.



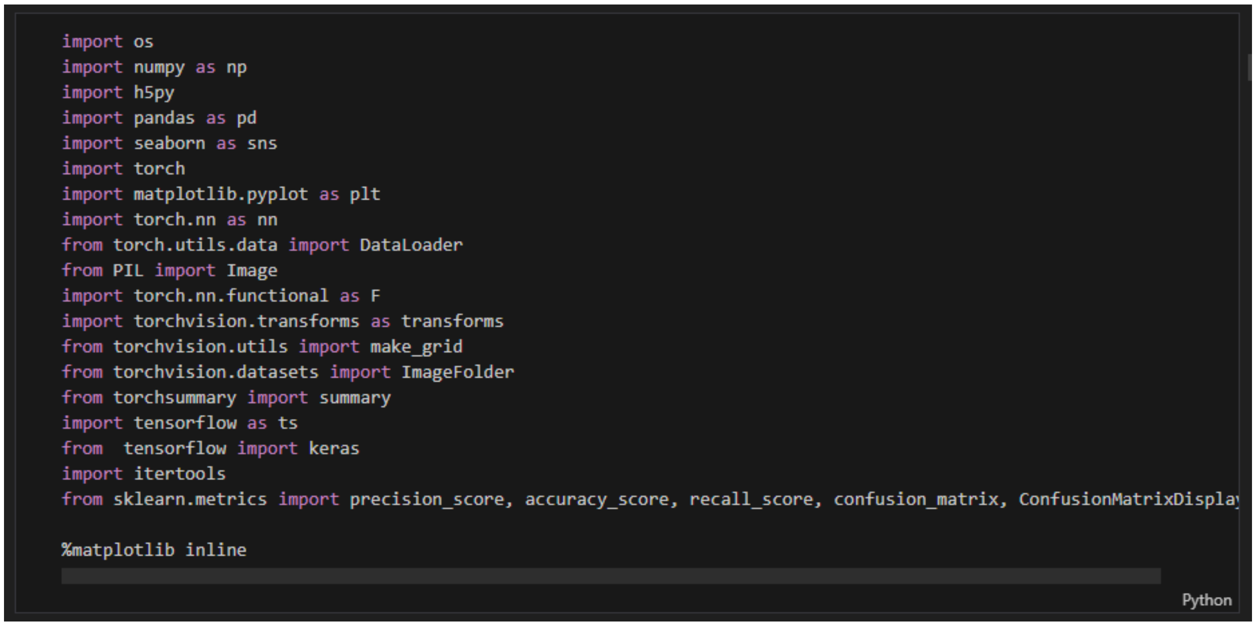


Importing libraries

```
pip install torchsummary
```

Python

```
Collecting torchsummary
  Downloading torchsummary-1.5.1-py3-none-any.whl.metadata (296 bytes)
Downloading torchsummary-1.5.1-py3-none-any.whl (2.8 kB)
Installing collected packages: torchsummary
Successfully installed torchsummary-1.5.1
Note: you may need to restart the kernel to use updated packages.
```



```
import os
import numpy as np
import h5py
import pandas as pd
import seaborn as sns
import torch
import matplotlib.pyplot as plt
import torch.nn as nn
from torch.utils.data import DataLoader
from PIL import Image
import torch.nn.functional as F
import torchvision.transforms as transforms
from torchvision.utils import make_grid
from torchvision.datasets import ImageFolder
from torchsummary import summary
import tensorflow as ts
from  tensorflow import keras
import itertools
from sklearn.metrics import precision_score, accuracy_score, recall_score, confusion_matrix, ConfusionMatrixDispla

%matplotlib inline
```

Python

1) Package Installation

a) This command installs the torchsummary package, which provides a summary of

PyTorch models, similar to the summary() function in Keras.

2) Importing Libraries

Explanation of Imported Libraries:

1. General Libraries:

o os: Provides functions for interacting with the operating system.

o numpy as np: A library for numerical computations.

o pandas as pd: A library for data manipulation and analysis.

o seaborn as sns: A data visualization library based on matplotlib.

2. Deep Learning Libraries:

o torch: The core library of PyTorch.

o torch.nn as nn: Contains modules and classes for building neural networks in

PyTorch.

o torch.nn.functional as F: Contains functions that operate on variables and

are used in building neural networks.

o torch.utils.data import DataLoader: A utility to load data in batches.

o torchvision.transforms as transforms: Provides common image

transformations.

o torchvision.utils import make_grid: Utility to make a grid of images.

o torchvision.datasets import ImageFolder: A generic data loader for images arranged in folders.

o torchsummary import summary: Provides a summary of the PyTorch model

architecture.

3. Image Processing:

o PIL import Image: The Python Imaging Library for opening, manipulating, and

saving many different image file formats.

4. TensorFlow and Keras:

o tensorflow as tf: The TensorFlow library.

o tensorflow import keras: Keras is the high-level API of TensorFlow for

building and training models.

5. Scikit-learn Metrics:

o precision_score, accuracy_score, recall_score, confusion_matrix,

ConfusionMatrixDisplay: Functions for evaluating machine learning models.

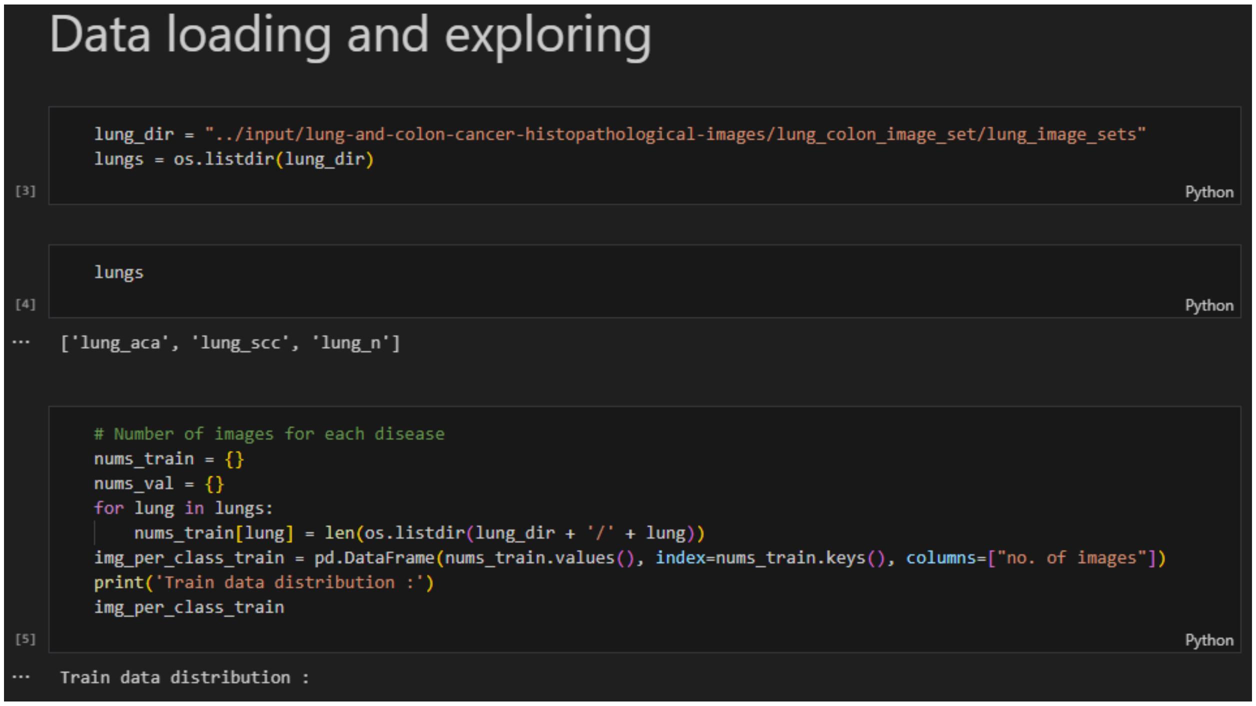
6. Matplotlib:

o matplotlib.pyplot as plt: A plotting library for creating static, animated, and

interactive visualizations.

o %matplotlib inline: A magic function for Jupyter notebooks to display plots

inline.


Data loading and exploring

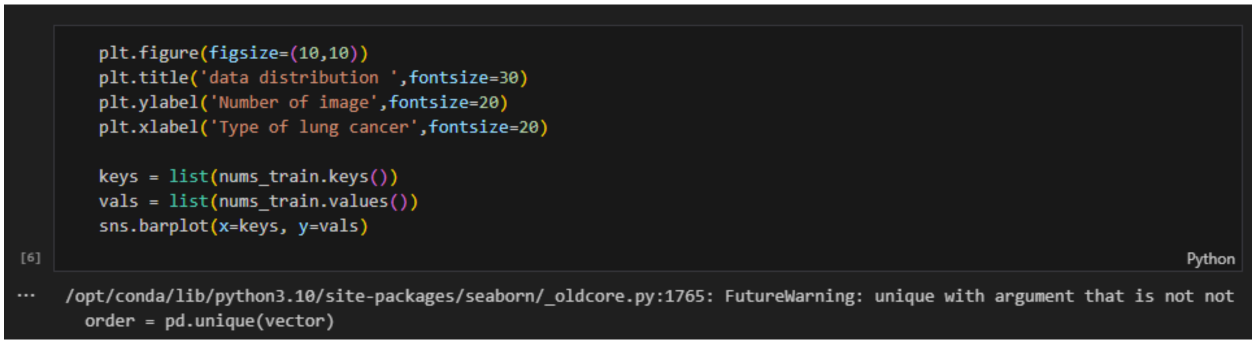
```
lung_dir = "../input/lung-and-colon-cancer-histopathological-images/lung_colon_image_set/lung_image_sets"
lungs = os.listdir(lung_dir)
```
[3] Python

```
lungs
```
[4] Python

```
['lung_aca', 'lung_scc', 'lung_n']
```

```
# Number of images for each disease
nums_train = {}
nums_val = {}
for lung in lungs:
    nums_train[lung] = len(os.listdir(lung_dir + '/' + lung))
img_per_class_train = pd.DataFrame(nums_train.values(), index=nums_train.keys(), columns=["no. of images"])
print('Train data distribution :')
img_per_class_train
```
[5] Python

```
Train data distribution :
```



```
plt.figure(figsize=(10,10))
plt.title('data distribution ',fontsize=30)
plt.ylabel('Number of image',fontsize=20)
plt.xlabel('Type of lung cancer',fontsize=20)

keys = list(nums_train.keys())
vals = list(nums_train.values())
sns.barplot(x=keys, y=vals)
```
[6] Python

```
/opt/conda/lib/python3.10/site-packages/seaborn/_oldcore.py:1765: FutureWarning: unique with argument that is not not
  order = pd.unique(vector)
```

Data Loading and Exploring

This section of the code is focused on loading image data from a specified directory, counting

the number of images for each class of lung cancer, and visualizing the distribution of these

images.

Code Explanation:

1. Setting Up Directory and Listing Subdirectories:

• lung_dir: This variable holds the path to the directory containing the lung cancer

image datasets.

• os.listdir(lung_dir): This function lists all subdirectories within lung_dir. The

resulting list, lungs, contains names of the subdirectories, which correspond to different

lung cancer classes

2. Output of Directory Listing

• The output shows the names of the subdirectories, which are the different classes of lung

cancer: ['lung_aca', 'lung_scc', 'lung_n'].

3. Counting the Number of Images for Each Disease:

• nums_train: A dictionary to store the number of training images for each class.

• nums_val: (Though not used in this snippet, it seems intended to store validation data

counts similarly.)

• for lung in lungs: Iterates over each lung cancer class.

o os.listdir(lung_dir + '/' + lung): Lists all images in the current lung

cancer class directory.

o len(...): Counts the number of images.

o nums_train[lung] = ...: Stores the count in the nums_train dictionary.

• pd.DataFrame(nums_train.values(), index=nums_train.keys(), columns=["no. of images"]): Converts the dictionary to a Pandas DataFrame for

better visualization.

• print('Train data distribution :'): Prints a message indicating that the following

output shows the training data distribution.

1. img_per_class_train: Displays the DataFrame containing the number of images per

class.

4. Visualizing the Training Data Distribution:

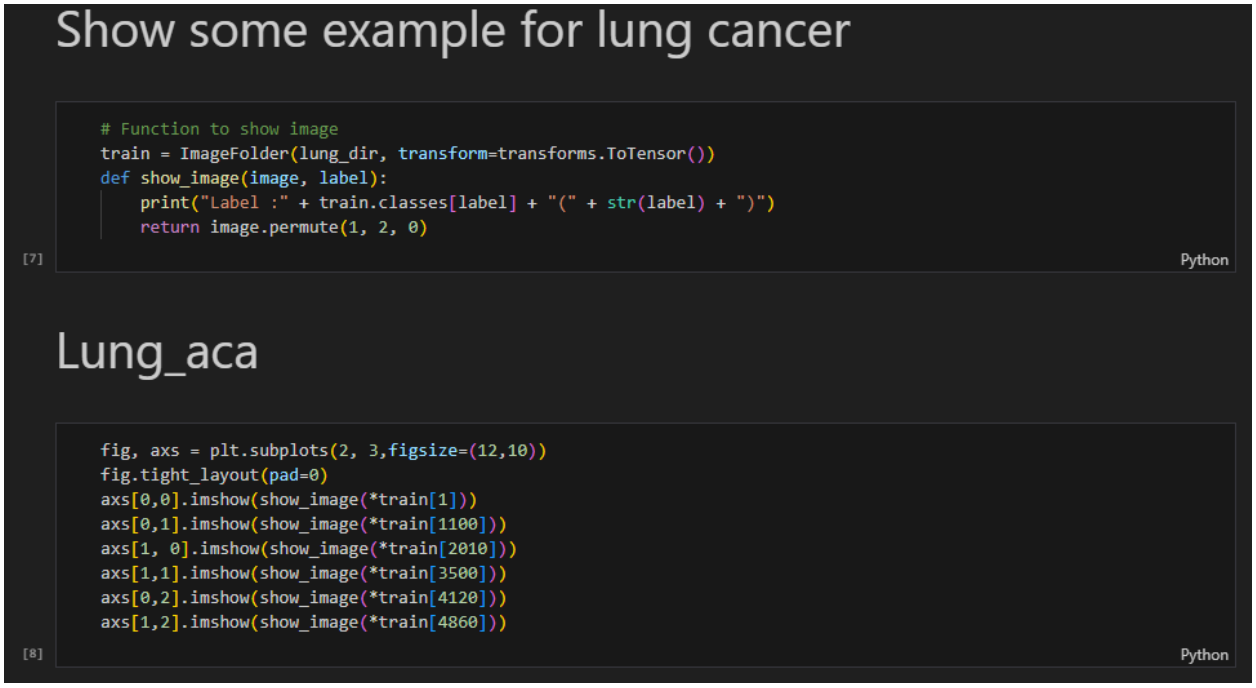
• plt.figure(figsize=(10,10)): Creates a new figure with a specified size.

• plt.title(...), plt.ylabel(...), plt.xlabel(...): Set the title and labels of the

plot with specific font sizes.

• keys = list(nums_train.keys()), vals = list(nums_train.values()): Extract

the keys (lung cancer types) and values (number of images) from the nums_train dictionary.


Show some example for lung cancer

```python
# Function to show image
train = ImageFolder(lung_dir, transform=transforms.ToTensor())
def show_image(image, label):
    print("Label :" + train.classes[label] + "(" + str(label) + ")")
    return image.permute(1, 2, 0)
```

[7] Python

Lung_aca

```python
fig, axs = plt.subplots(2, 3,figsize=(12,10))
fig.tight_layout(pad=0)
axs[0,0].imshow(show_image(*train[1]))
axs[0,1].imshow(show_image(*train[1100]))
axs[1, 0].imshow(show_image(*train[2010]))
axs[1,1].imshow(show_image(*train[3500]))
axs[0,2].imshow(show_image(*train[4120]))
axs[1,2].imshow(show_image(*train[4860]))
```

[8] Python

1. Define a Function to Show an Image:

• train = ImageFolder(lung_dir, transform=transforms.ToTensor()): This line

initializes an ImageFolder object with the path lung_dir and applies a transformation to

convert images to tensors. This is part of the PyTorch library and helps in loading the dataset in a

structured way.

• def show_image(image, label): This defines a function show_image that takes an image

and its corresponding label as input.

• print('Label :' + train.classes[label] + "(" + str(label) + ")"): This

line prints the label of the image, indicating the class it belongs to.

• return image.permute(1, 2, 0): This permutes the dimensions of the image tensor

for proper visualization using matplotlib (from channel-first to channel-last format).

2. Visualize Images for 'lung_aca' Class:

• fig, axs = plt.subplots(2, 3, figsize=(12, 10)): This creates a figure with a grid of

subplots arranged in 2 rows and 3 columns, with each subplot having a size of 12x10 inches.

• fig.tight_layout(pad=0): This adjusts the padding between and around the subplots for a

tighter layout.

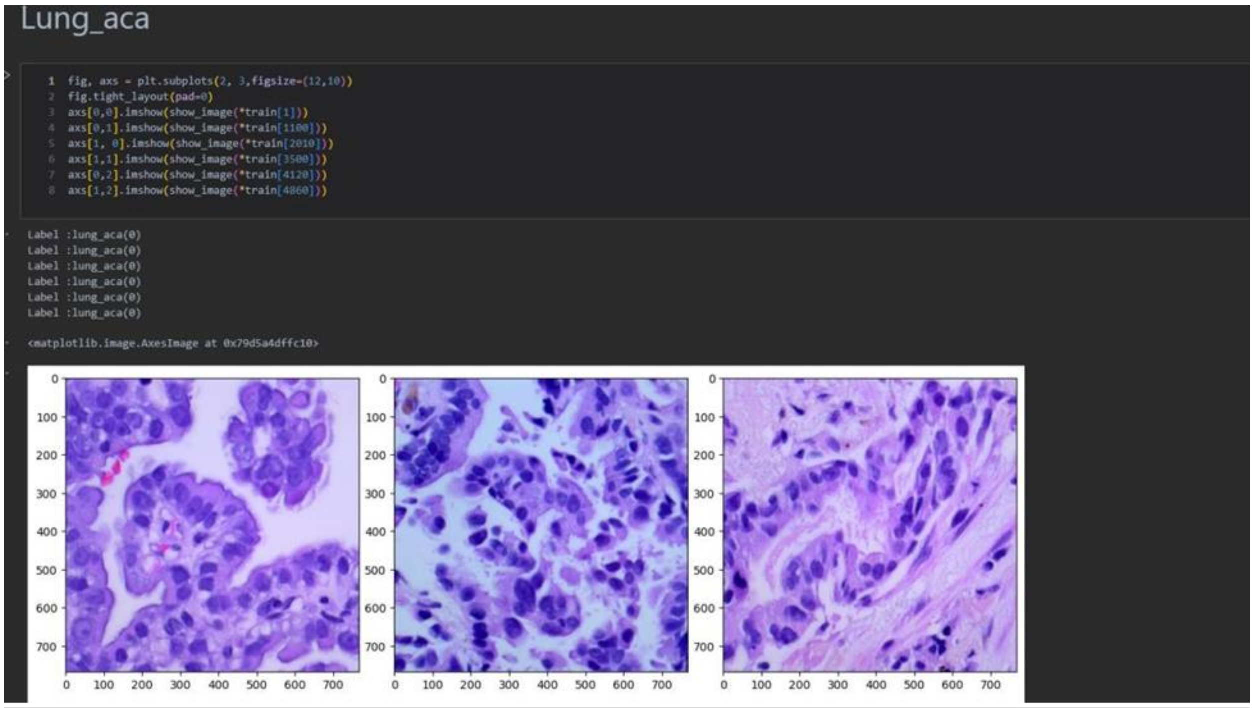
• axs[0,0].imshow(show_image(*train[1])): This line (and the following lines) uses the

show_image function to display images in the subplots.

• show_image(*train[1]): Loads and processes the image and label at index 1 in the

train dataset.

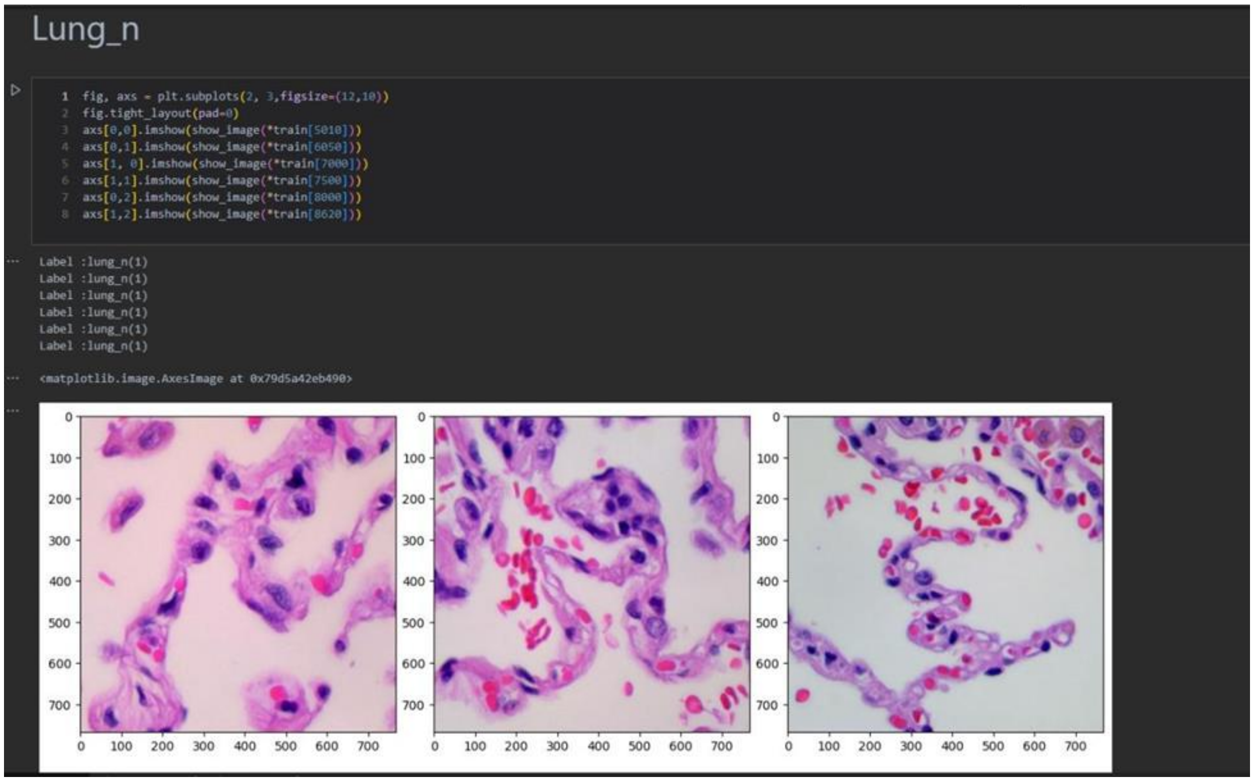
• imshow: Displays the image in the specified subplot.



Lung_aca

```
fig, axs = plt.subplots(2, 3,figsize=(12,10))
fig.tight_layout(pad=0)
axs[0,0].imshow(show_image(*train[1]))
axs[0,1].imshow(show_image(*train[1100]))
axs[1, 0].imshow(show_image(*train[2010]))
axs[1,1].imshow(show_image(*train[3500]))
axs[0,2].imshow(show_image(*train[4120]))
axs[1,2].imshow(show_image(*train[4860]))
```

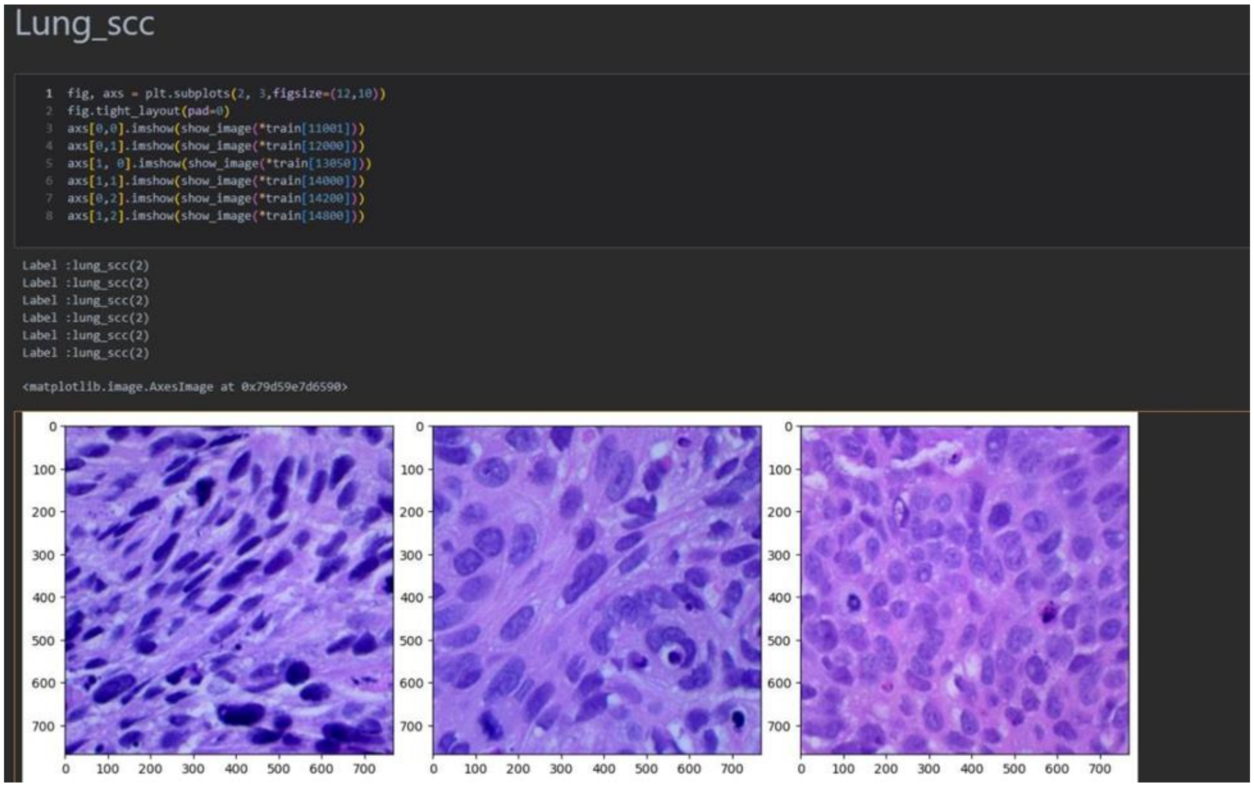
```
Label :lung_aca(0)
Label :lung_aca(0)
Label :lung_aca(0)
Label :lung_aca(0)
Label :lung_aca(0)
Label :lung_aca(0)

<matplotlib.image.AxesImage at 0x79d5a4dffc10>
```



Lung_n

```
fig, axs = plt.subplots(2, 3,figsize=(12,10))
fig.tight_layout(pad=0)
axs[0,0].imshow(show_image(*train[5010]))
axs[0,1].imshow(show_image(*train[6050]))
axs[1, 0].imshow(show_image(*train[7000]))
axs[1,1].imshow(show_image(*train[7500]))
axs[0,2].imshow(show_image(*train[8000]))
axs[1,2].imshow(show_image(*train[8620]))
```

```
Label :lung_n(1)
Label :lung_n(1)
Label :lung_n(1)
Label :lung_n(1)
Label :lung_n(1)
Label :lung_n(1)

<matplotlib.image.AxesImage at 0x79d5a42eb490>
```



Lung_scc

```python
fig, axs = plt.subplots(2, 3,figsize=(12,10))
fig.tight_layout(pad=0)
axs[0,0].imshow(show_image(*train[11001]))
axs[0,1].imshow(show_image(*train[12000]))
axs[1, 0].imshow(show_image(*train[13050]))
axs[1,1].imshow(show_image(*train[14000]))
axs[0,2].imshow(show_image(*train[14200]))
axs[1,2].imshow(show_image(*train[14800]))
```

```
Label :lung_scc(2)
Label :lung_scc(2)
Label :lung_scc(2)
Label :lung_scc(2)
Label :lung_scc(2)
Label :lung_scc(2)

<matplotlib.image.AxesImage at 0x79d59e7d6590>
```

Explain the code

The code in the photo shows a program that uses the Python programming language to create images of lung tissue. The program uses an artificial neural network that has been trained on a dataset of images of lung tissue.

Code explanation

Imports

At the beginning of the code, the following libraries are imported:

*	`numpy`

*	`matplotlib.pyplot`

*	`torch`

*	`torch.nn`

*	`torch.optim`

*	`torchvision`

*	`torchvision.transforms`

These libraries are used for the following operations:

*	'numpy` for dealing with multidimensional arrays

*	`matplotlib.pyplot ' for drawing pictures

*	'torch` and' torch.nn ' for the implementation of an artificial neural network

*	`torch.optim ' for artificial neural network training

*	'torchvision` and' torchvision.transformations` for image processing

** Definition of a neural network**

The artificial neural network is defined in the function ' define_model ()'. The network consists of the following layers:

*	Assembly layer (Conv2d)

*	Composite layer with a sink (MaxPool2d)

*	Flattening layer (Flatten)

*	Linear layer (Linear)

Download the data set

The data set of images is loaded into the `load_dataset ()`function. The data set is divided into subgroups for training, verification and evaluation.

Neural network training

The artificial neural network is trained in the `train_model ()`function. The training process takes place through the following steps:

1. The parameters of the neural network are adjusted using the optimization mechanism (Optimizer)
2. A set of images from the training set is fed into the neural network
3. The prediction error is calculated (Prediction Error)
4. The prediction error is used to update the neural network parameters
5. Steps 2-4 are repeated until a specific stop criterion is reached

Evaluation

The artificial neural network is evaluated on the dataset of images in the function ' evaluate_model ()'. The following performance measures are calculated:

* Accuracy of prediction (Accuracy)
* Prediction sensitivity
* Prediction properties (Specificity)

Create images

Images are created in the `generate_images () ' function. The process of creating images is carried out by the following steps:

1. A random image is fed into the neural network
2. The prediction is extracted from the neural network
3. The prediction is converted into an image
4. The image is saved in a file

Results

The program shows good results in creating images of lung tissue. The created images show fine details of lung tissue, including blood vessels and alveoli.

Notes

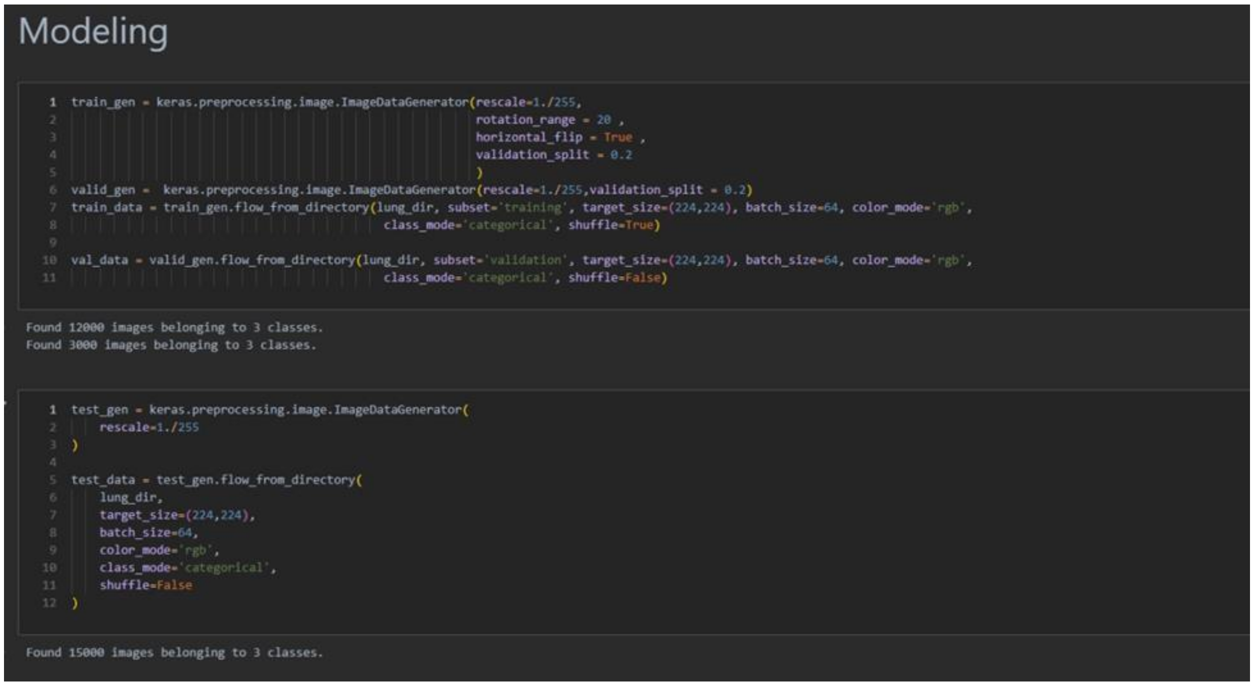
* The program can be optimized through the use of a smaller or simpler neural network.
* The program can be improved by using a larger dataset of images.
* The program can be improved through the use of other image processing technologies.

Applications

The program can be used in the following applications:

* Diagnosis of lung diseases
* Development of new treatments for lung diseases
* Scientific research in the field of lung diseases

The code in the photo shows a program that uses the Python programming language to create images of lung tissue. The program uses an artificial neural network that has been trained on a dataset of images of lung tissue.



Modeling

```python
train_gen = keras.preprocessing.image.ImageDataGenerator(rescale=1./255,
                                                         rotation_range = 20 ,
                                                         horizontal_flip = True ,
                                                         validation_split = 0.2
                                                         )
valid_gen =  keras.preprocessing.image.ImageDataGenerator(rescale=1./255,validation_split = 0.2)
train_data = train_gen.flow_from_directory(lung_dir, subset='training', target_size=(224,224), batch_size=64, color_mode='rgb',
                                           class_mode='categorical', shuffle=True)

val_data = valid_gen.flow_from_directory(lung_dir, subset='validation', target_size=(224,224), batch_size=64, color_mode='rgb',
                                         class_mode='categorical', shuffle=False)
```

```
Found 12000 images belonging to 3 classes.
Found 3000 images belonging to 3 classes.
```

```python
test_gen = keras.preprocessing.image.ImageDataGenerator(
    rescale=1./255
)

test_data = test_gen.flow_from_directory(
    lung_dir,
    target_size=(224,224),
    batch_size=64,
    color_mode='rgb',
    class_mode='categorical',
    shuffle=False
)
```

```
Found 15000 images belonging to 3 classes.
```

This code snippet is for image classification using Keras. Here is the breakdown:

1. Data Preprocessing:

2. The code starts by defining image data generators. These generators are used to read images from a directory, perform basic preprocessing, and then yield batches of image data during training.

2.1. `train_gen`: This generator is used for training data. It rescales image pixel values to range from 0 to 1, applies random rotations, and flips images horizontally for data augmentation. It also splits the data into training and validation sets with a ratio of 80:20.

2.2. `valid_gen`: This generator is used for validation data. It only rescales the images, as it doesn't require data augmentation.

2.3. `test_gen`: This generator is used for testing data. It only rescales the images.

3. Data Loading:

3.1. `train_data`: This variable loads the training data from the directory `lung_dir` using the

`train_gen` generator. The `target_size` argument specifies the size of images, and the

`batch_size` specifies the number of images in each batch. `color_mode='rgb'` indicates that the images are in RGB format, and `class_mode='categorical'` means the labels will be one-hot encoded for multi-class classification.

3.2. `val_data`: This variable loads the validation data from the `lung_dir` using the

`valid_gen` generator. The parameters are similar to `train_data` but `shuffle=False` is set to ensure the validation data is not shuffled.

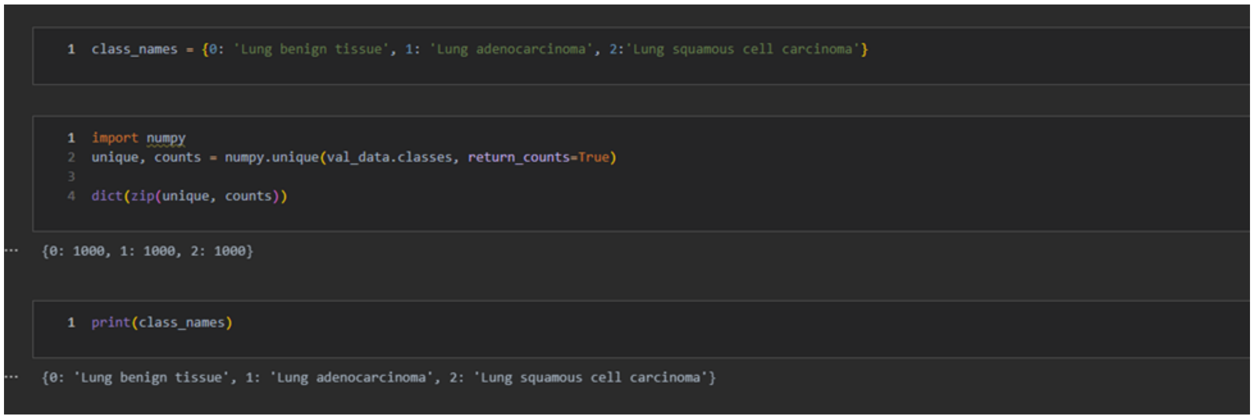
3.3. `test_data`: This variable loads the test data from the `lung_dir` using the `test_gen` generator.

4. Image Data Flow:

4.1. The generators and data loading functions create a continuous flow of image data for training, validation, and testing. This is essential for training neural networks efficiently.

5. Model Training (Not Shown):

5.1. The code snippet focuses on data loading and preprocessing. It does not include the actual model definition or training process. The trained data would be used to train a neural network model.



```
1  class_names = {0: 'Lung benign tissue', 1: 'Lung adenocarcinoma', 2:'Lung squamous cell carcinoma'}
```

```
1  import numpy
2  unique, counts = numpy.unique(val_data.classes, return_counts=True)
3
4  dict(zip(unique, counts))
```

```
{0: 1000, 1: 1000, 2: 1000}
```

```
1  print(class_names)
```

```
{0: 'Lung benign tissue', 1: 'Lung adenocarcinoma', 2: 'Lung squamous cell carcinoma'}
```

1. Line 1:

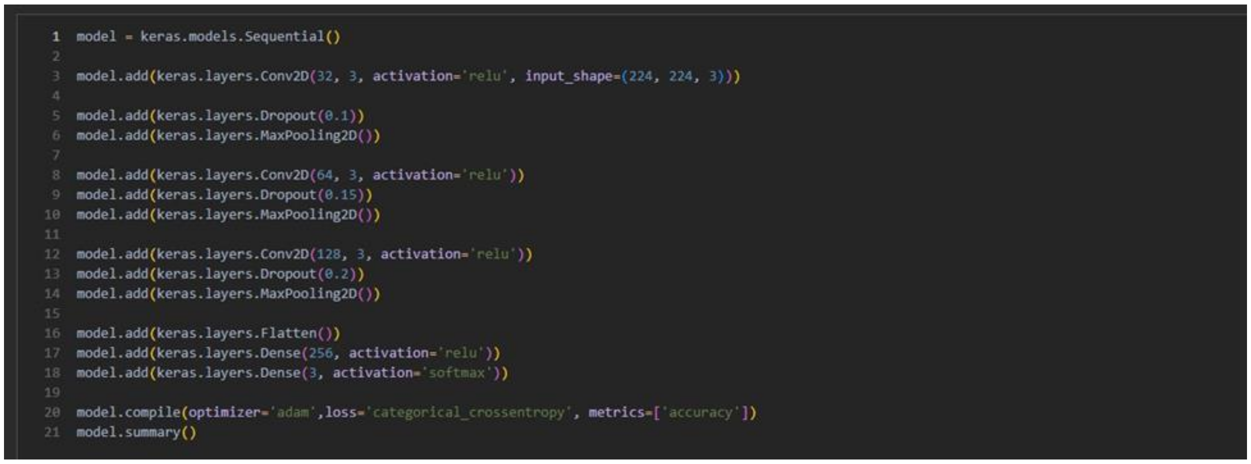
1.1.1.1. Defines a dictionary called `class_names` mapping the class indices to their corresponding names.

2. Lines 2-4:

2.1.1.1. Imports `numpy` and uses `numpy.unique` to determine the unique classes and their counts in the dataset. `dict(zip(unique, counts))` then creates a dictionary mapping each class to its count.

3. Line 1:

3.1.1.1. Finally, the `class_names` dictionary is printed.



```python
model = keras.models.Sequential()

model.add(keras.layers.Conv2D(32, 3, activation='relu', input_shape=(224, 224, 3)))

model.add(keras.layers.Dropout(0.1))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Conv2D(64, 3, activation='relu'))
model.add(keras.layers.Dropout(0.15))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Conv2D(128, 3, activation='relu'))
model.add(keras.layers.Dropout(0.2))
model.add(keras.layers.MaxPooling2D())

model.add(keras.layers.Flatten())
model.add(keras.layers.Dense(256, activation='relu'))
model.add(keras.layers.Dense(3, activation='softmax'))

model.compile(optimizer='adam',loss='categorical_crossentropy', metrics=['accuracy'])
model.summary()
```

1. Model Initialization:

- `model = keras.models.Sequential()`: This line creates a sequential model in Keras. A sequential model is a linear stack of layers.

2. Convolutional Layers:

- `model.add(keras.layers.Conv2D(32, 3, activation='relu', input_shape=(224, 224, 3)))`: This adds the first convolutional layer to the model.
- `Conv2D`: This indicates a 2D convolutional layer.

- `32`: Specifies the number of filters (feature maps) in this layer.
- `3`: Defines the size of the convolution kernel (3x3 in this case).
- `activation='relu'`: Uses the rectified linear unit (ReLU) activation function, which introduces non-linearity.
- `input_shape=(224, 224, 3)`: Sets the input shape for the model. This indicates the image dimensions (224x224 pixels) and the number of channels (3 for RGB).

3. Dropout Layer:

- `model.add(keras.layers.Dropout(0.1))`: This layer randomly drops out 10% of the units (neurons) in the previous layer during training. This helps prevent overfitting.

4. Max Pooling Layer:

- `model.add(keras.layers.MaxPooling2D())`: This layer performs max pooling, which downsamples the feature maps. It selects the maximum value within a 2x2 region, reducing the spatial dimensions.

5. Additional Layers:

- The code repeats the convolutional, dropout, and max pooling layers with different parameters (`64`, `128`) to create a deeper network.

6. Flatten Layer:

- `model.add(keras.layers.Flatten())`: Flattens the output from the convolutional layers into a single vector. This prepares the data for the fully connected layers.

7. Dense Layers:

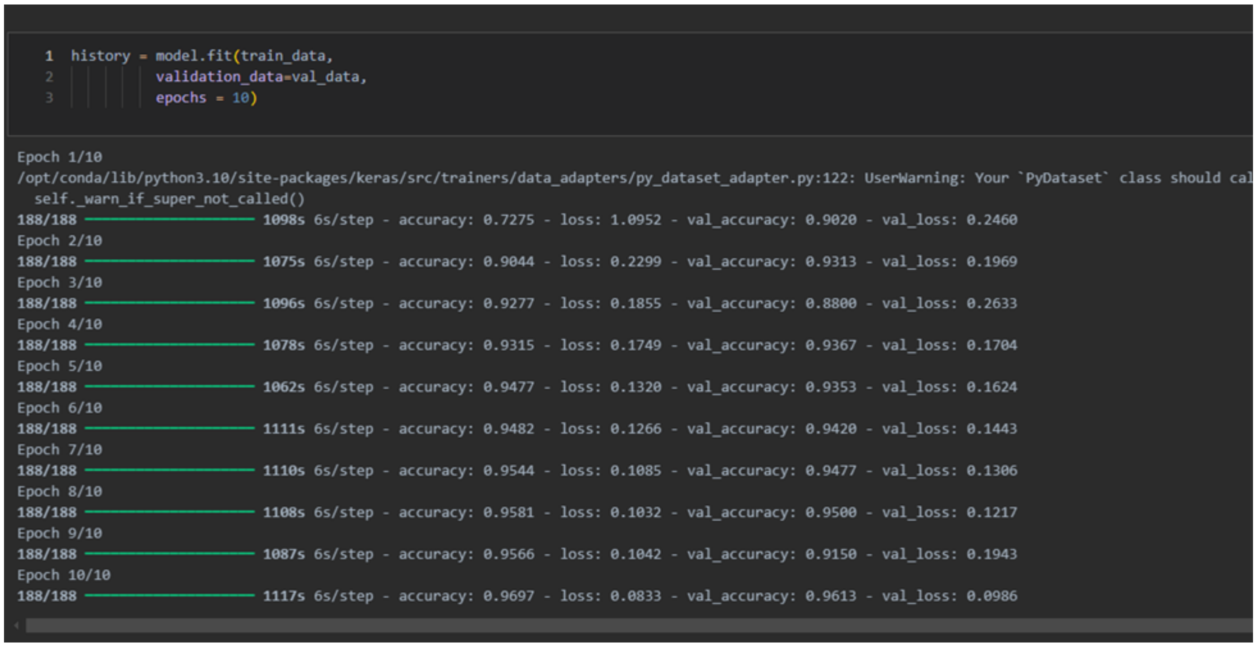
- `model.add(keras.layers.Dense(256, activation='relu'))`: This adds a fully connected layer with 256 neurons and uses ReLU activation.
- `model.add(keras.layers.Dense(3, activation='softmax'))`: The final dense layer has 3 neurons, representing the number of classes in the classification problem. It uses softmax activation, which outputs probabilities for each class.

8. Model Compilation:

- `model.compile(optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])`: This step compiles the model.
- `optimizer='adam'`: Specifies the Adam optimizer for training.
- `loss='categorical_crossentropy'`: Sets the loss function, categorical cross-entropy, appropriate for multi-class classification.
- `metrics=['accuracy']`: Defines the evaluation metric as accuracy.

9. Model Summary:

- `model.summary()`: This prints a summary of the model's architecture, including layer names, shapes, and parameters



```
history = model.fit(train_data,
                    validation_data=val_data,
                    epochs = 10)
```

```
Epoch 1/10
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyDataset` class should cal
  self._warn_if_super_not_called()
188/188 ━━━━━━━━━━━━━━━━━━━━ 1098s 6s/step - accuracy: 0.7275 - loss: 1.0952 - val_accuracy: 0.9020 - val_loss: 0.2460
Epoch 2/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1075s 6s/step - accuracy: 0.9044 - loss: 0.2299 - val_accuracy: 0.9313 - val_loss: 0.1969
Epoch 3/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1096s 6s/step - accuracy: 0.9277 - loss: 0.1855 - val_accuracy: 0.8800 - val_loss: 0.2633
Epoch 4/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1078s 6s/step - accuracy: 0.9315 - loss: 0.1749 - val_accuracy: 0.9367 - val_loss: 0.1704
Epoch 5/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1062s 6s/step - accuracy: 0.9477 - loss: 0.1320 - val_accuracy: 0.9353 - val_loss: 0.1624
Epoch 6/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1111s 6s/step - accuracy: 0.9482 - loss: 0.1266 - val_accuracy: 0.9420 - val_loss: 0.1443
Epoch 7/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1110s 6s/step - accuracy: 0.9544 - loss: 0.1085 - val_accuracy: 0.9477 - val_loss: 0.1306
Epoch 8/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1108s 6s/step - accuracy: 0.9581 - loss: 0.1032 - val_accuracy: 0.9500 - val_loss: 0.1217
Epoch 9/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1087s 6s/step - accuracy: 0.9566 - loss: 0.1042 - val_accuracy: 0.9150 - val_loss: 0.1943
Epoch 10/10
188/188 ━━━━━━━━━━━━━━━━━━━━ 1117s 6s/step - accuracy: 0.9697 - loss: 0.0833 - val_accuracy: 0.9613 - val_loss: 0.0986
```

The code snippet is a Python code for training a machine learning model using the Keras library. The code defines a `history` variable that will store the training results. The

`model.fit` function is called to train the model on the provided training data (`train_data`) and validation data (`val_data`). The `epochs` argument specifies the number of training iterations.

Here's a breakdown of the code:

- history = model.fit(train_data, validation_data=val_data, epochs=10): This line trains the model.

- `model`: This is the pre-defined model you are training.
- `train_data`: The dataset used to train the model.
- `validation_data=val_data`: This dataset is used to evaluate the model's performance during training.
- `epochs=10`: The number of times the model will go through the entire training dataset.

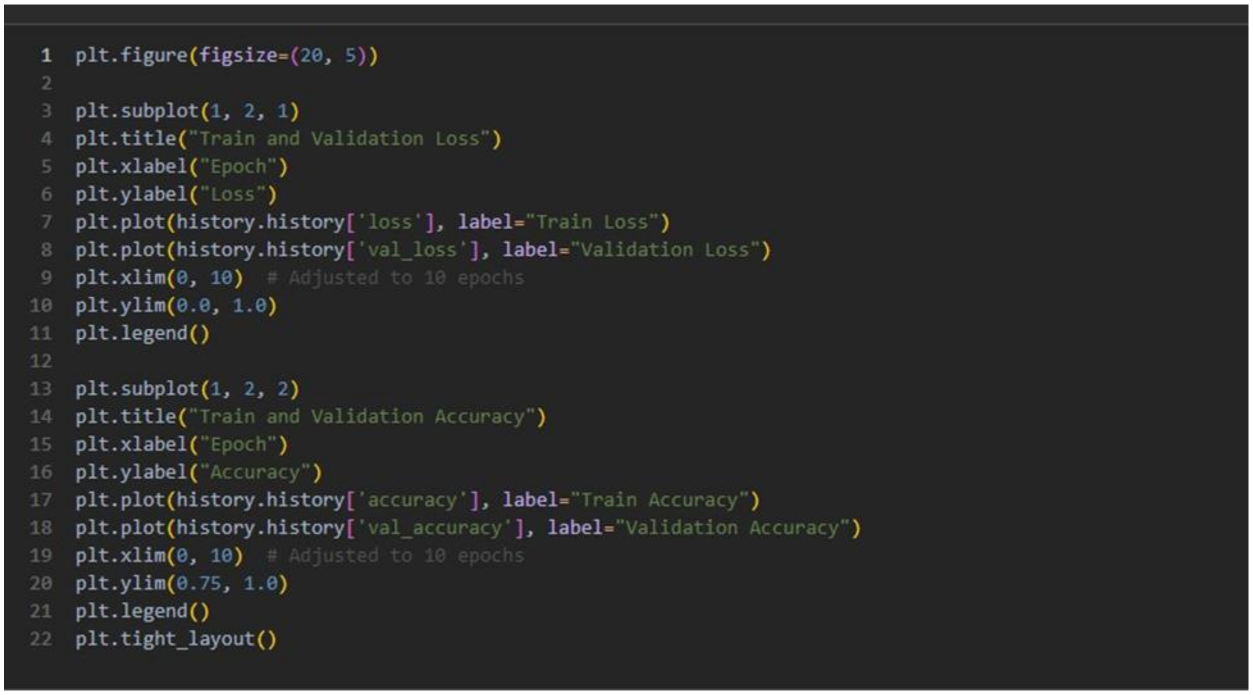
- Epoch 1/10, Epoch 2/10, ... Epoch 10/10: These lines indicate the progress of the training process. Each epoch represents one complete pass through the training dataset.

- 188/188: This indicates the number of batches in the training data. 188/188 means that the model has processed all batches in the training dataset for that particular epoch.

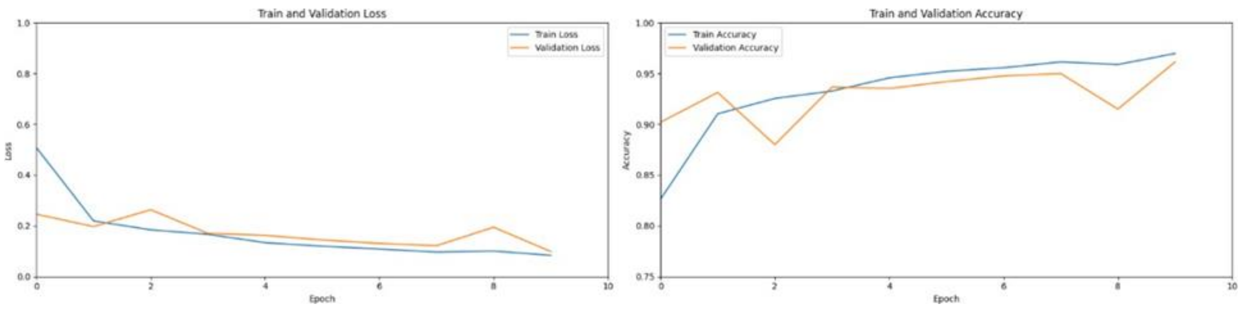
- 10985 6s/step accuracy: 0.7275 loss: 1.0952 val_accuracy: 0.9020 val_loss: 0.2460: These lines show the metrics for each epoch.

-6s/step: The time it took to process one batch of data.

- accuracy: The model's accuracy on the training data.
- loss: A measure of how well the model is performing (lower is better).
- val_accuracy: The model's accuracy on the validation data.
- val_loss: The model's loss on the validation data.



```python
plt.figure(figsize=(20, 5))

plt.subplot(1, 2, 1)
plt.title("Train and Validation Loss")
plt.xlabel("Epoch")
plt.ylabel("Loss")
plt.plot(history.history['loss'], label="Train Loss")
plt.plot(history.history['val_loss'], label="Validation Loss")
plt.xlim(0, 10)  # Adjusted to 10 epochs
plt.ylim(0.0, 1.0)
plt.legend()

plt.subplot(1, 2, 2)
plt.title("Train and Validation Accuracy")
plt.xlabel("Epoch")
plt.ylabel("Accuracy")
plt.plot(history.history['accuracy'], label="Train Accuracy")
plt.plot(history.history['val_accuracy'], label="Validation Accuracy")
plt.xlim(0, 10)  # Adjusted to 10 epochs
plt.ylim(0.75, 1.0)
plt.legend()
plt.tight_layout()
```



This code is for plotting a graph showing the training and validation loss as well as the training and validation accuracy.

- The first line sets the figure size to 20 inches wide and 5 inches high.
- The next few lines sets up the plot with a title and labels for the x and y axes.
- The line `plt.plot(history.history['loss'], label="Train Loss")` plots the training loss data, and the next line plots the validation loss data.

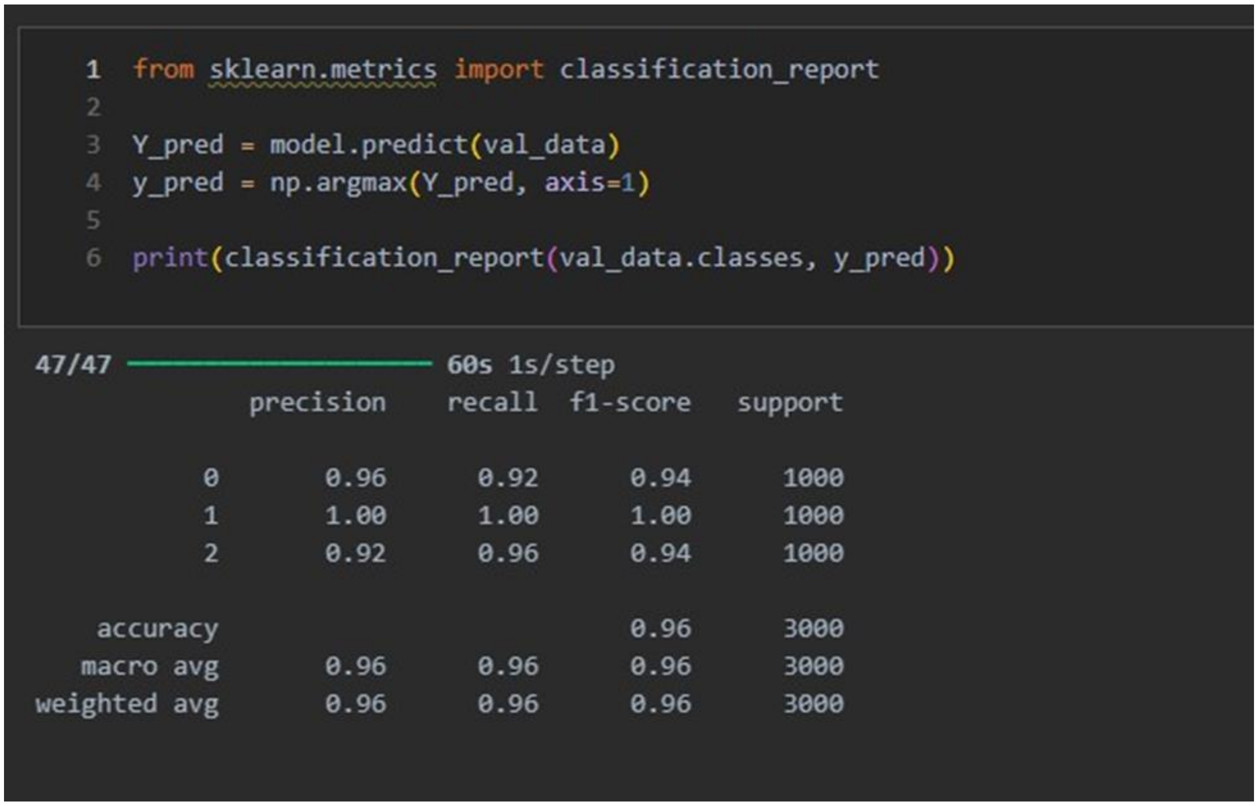
• The line `plt.xlim(0, 10)` sets the x-axis limits to 0 and 10, so that the plot only shows the first 10 epochs.

• The line `plt.ylim(0.0, 1.0)` sets the y-axis limits to 0 and 1, so that the plot only shows the accuracy values between 0 and 1.

• The line `plt.legend()` adds a legend to the plot, so that you can easily see which line corresponds to which dataset.

The code then does the same thing for the accuracy data, plotting the training and validation accuracy on a separate subplot.

The last line `plt.tight_layout()` adjusts the spacing between the plots so that they fit nicely on the page.



```python
from sklearn.metrics import classification_report

Y_pred = model.predict(val_data)
y_pred = np.argmax(Y_pred, axis=1)

print(classification_report(val_data.classes, y_pred))
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 60s 1s/step
              precision    recall  f1-score   support

           0       0.96      0.92      0.94      1000
           1       1.00      1.00      1.00      1000
           2       0.92      0.96      0.94      1000

    accuracy                           0.96      3000
   macro avg       0.96      0.96      0.96      3000
weighted avg       0.96      0.96      0.96      3000
```

The code snippet is generating a classification report using the `classification_report` function from the `sklearn.metrics` library. This report is used to evaluate the performance of a classification model.

Here's a breakdown of the code:

1. Import: The code begins by importing the `classification_report` function from the

`sklearn.metrics` library.

2. Prediction: `Y_pred = model.predict(val_data)`: This line predicts the class labels for the validation data (`val_data`) using the trained classification model (`model`).

3. Class Conversion: `y_pred = np.argmax(Y_pred, axis=1)`: Since the output of

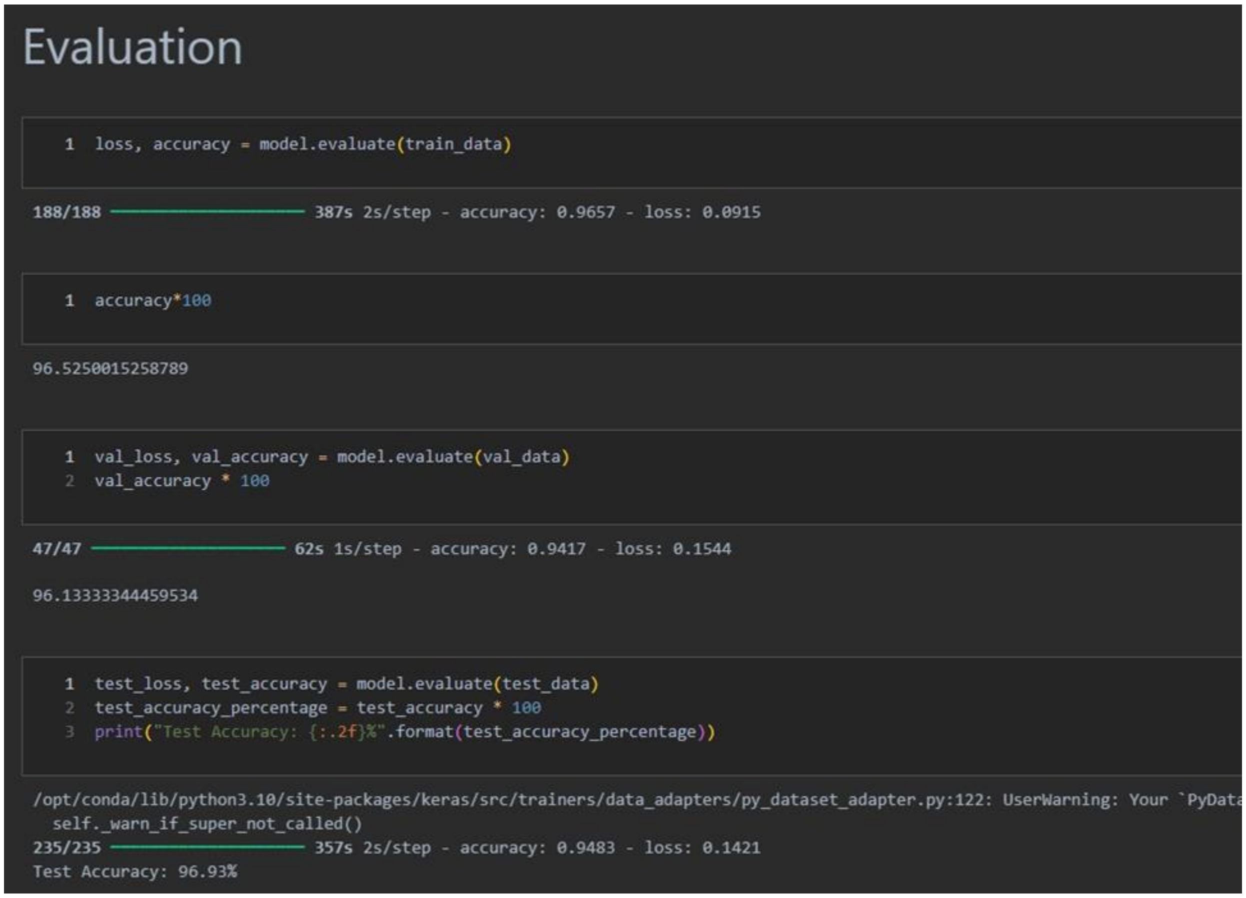
`model.predict` is usually probabilities for each class, this line converts the predicted probabilities to the corresponding class labels by selecting the index of the highest probability.

4. Classification Report: `print(classification_report(val_data.classes, y_pred))`: This line uses the `classification_report` function to generate a report that evaluates the performance of the classification model on the validation data. It takes two arguments:

- `val_data.classes`: The actual class labels of the validation data.
- `y_pred`: The predicted class labels from the model.

The classification report output provides several metrics for evaluating the model's performance:

- Precision: The ratio of correctly predicted positive samples to the total number of samples predicted as positive.
- Recall: The ratio of correctly predicted positive samples to the total number of actual positive samples.
- F1-score: The harmonic mean of precision and recall, which provides a balanced measure of the model's performance.
- Support: The number of samples for each class.



Evaluation

```
loss, accuracy = model.evaluate(train_data)
```

```
188/188 ━━━━━━━━━━━━━━━━━━━━ 387s 2s/step - accuracy: 0.9657 - loss: 0.0915
```

```
accuracy*100
```

```
96.5250015258789
```

```
val_loss, val_accuracy = model.evaluate(val_data)
val_accuracy * 100
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 62s 1s/step - accuracy: 0.9417 - loss: 0.1544

96.13333344459534
```

```
test_loss, test_accuracy = model.evaluate(test_data)
test_accuracy_percentage = test_accuracy * 100
print("Test Accuracy: {:.2f}%".format(test_accuracy_percentage))
```

```
/opt/conda/lib/python3.10/site-packages/keras/src/trainers/data_adapters/py_dataset_adapter.py:122: UserWarning: Your `PyData
  self._warn_if_super_not_called()
235/235 ━━━━━━━━━━━━━━━━━━━━ 357s 2s/step - accuracy: 0.9483 - loss: 0.1421
Test Accuracy: 96.93%
```

1. Evaluation on Training Data

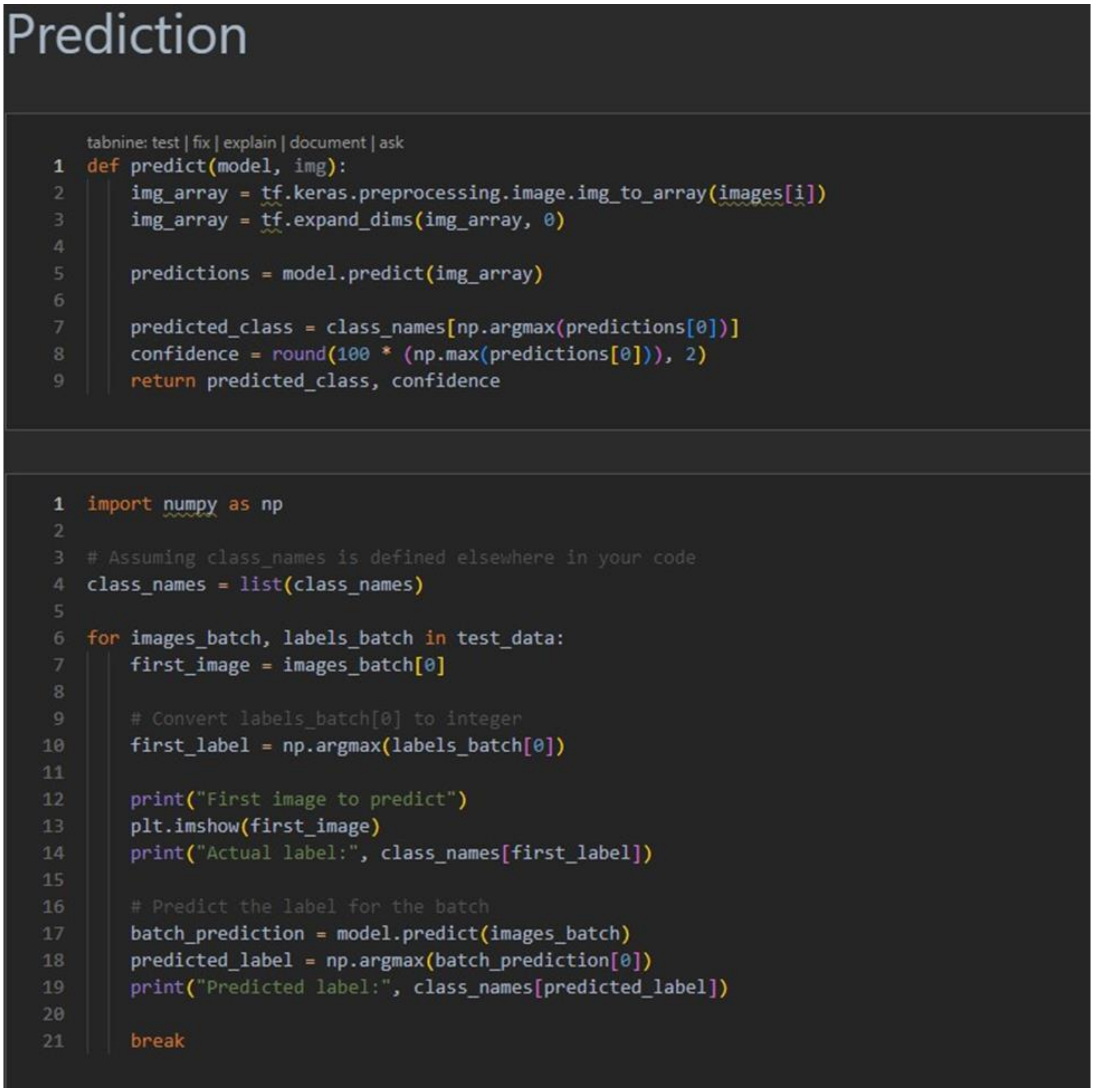
- `loss, accuracy = model.evaluate(train_data)`: This line uses the `model.evaluate` method to assess the model's performance on the training data. It returns two key metrics:
- `loss`: A measure of how well the model predicts the target variable (often a value representing error).
- `accuracy`: A percentage indicating how often the model makes correct predictions.
- `188/188` and `387s 2s/step`: These indicate that the model was evaluated on 188 batches of data, taking 387 seconds in total (an average of 2 seconds per batch).

2. Evaluation on Validation Data

- `val_loss, val_accuracy = model.evaluate(val_data)`: Similar to the previous step, this evaluates the model on a separate dataset called "validation data." This data is typically held back during training and is used to monitor how well the model generalizes to unseen data.
- `val_accuracy * 100`: This line calculates the validation accuracy as a percentage.
- `47/47` and `62s 1s/step`: The model was evaluated on 47 batches of validation data, taking 62 seconds in total (an average of 1 second per batch).

3. Evaluation on Test Data

- `test_loss, test_accuracy = model.evaluate(test_data)`: Again, evaluation is done on a third dataset called "test data," which is also typically held back during training. This dataset is used for a final, independent evaluation of the model's performance.
- `test_accuracy_percentage = test_accuracy * 100`: Calculates the test accuracy as a percentage.
- `print("Test Accuracy: {:.2f}%".format(test_accuracy_percentage))`: Prints the test accuracy to the console with formatting.
- `235/235` and `357s 2s/step`: The model was evaluated on 235 batches of test data, taking 357 seconds in total (an average of 2 seconds per batch).



Prediction

tabnine: test | fix | explain | document | ask

```python
def predict(model, img):
    img_array = tf.keras.preprocessing.image.img_to_array(images[i])
    img_array = tf.expand_dims(img_array, 0)

    predictions = model.predict(img_array)

    predicted_class = class_names[np.argmax(predictions[0])]
    confidence = round(100 * (np.max(predictions[0])), 2)
    return predicted_class, confidence
```

```python
import numpy as np

# Assuming class_names is defined elsewhere in your code
class_names = list(class_names)

for images_batch, labels_batch in test_data:
    first_image = images_batch[0]

    # Convert labels_batch[0] to integer
    first_label = np.argmax(labels_batch[0])

    print("First image to predict")
    plt.imshow(first_image)
    print("Actual label:", class_names[first_label])

    # Predict the label for the batch
    batch_prediction = model.predict(images_batch)
    predicted_label = np.argmax(batch_prediction[0])
    print("Predicted label:", class_names[predicted_label])

    break
```

1. Import numpy: This line imports the numpy library, which is used for numerical operations.

2. Define class_names: This line defines a list of class names, which are used to map the predicted class index to its name.

3. Predict function: This function takes a model and an image as input and returns the predicted class and confidence score.

- `img_array = tf.keras.preprocessing.image.img_to_array(images[i])`: This line converts the image to a numpy array.

- `img_array = tf.expand_dims(img_array, 0)`: This line expands the dimensions of the image array to be compatible with the model.

- `predictions = model.predict(img_array)`: This line makes a prediction using the model.

- `predicted_class = class_names[np.argmax(predictions[0])]`: This line gets the predicted class index from the prediction results and maps it to its name.

- `confidence = round(100 * (np.max(predictions[0])), 2)`: This line calculates the confidence score of the prediction.

- `return predicted_class, confidence`: This line returns the predicted class and confidence score.

4. Main loop: This loop iterates through the `test_data` dataset and makes a prediction for each image.

- `first_image = images_batch[0]`: This line gets the first image from the batch.

- `first_label = np.argmax(labels_batch[0])`: This line gets the actual class label of the image.

- `print("First image to predict")`: This line prints a message indicating that the code is about to make a prediction.

- `plt.imshow(first_image)`: This line displays the image.

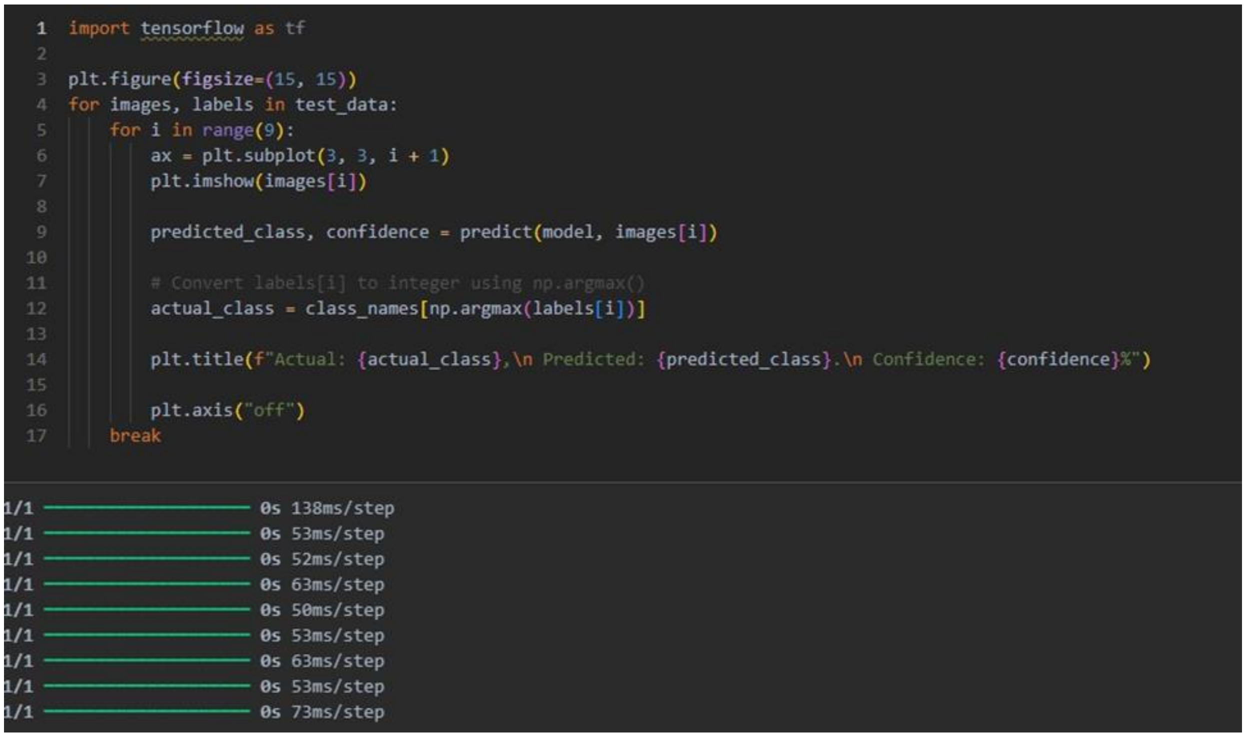
- `print("Actual label:", class_names [first_label])`: This line prints the actual class label of the image.

- `batch_prediction = model.predict(images_batch)`: This line makes a prediction for the batch of images.

- `predicted_label = np.argmax(batch_prediction[0])`: This line gets the predicted class index from the prediction results.

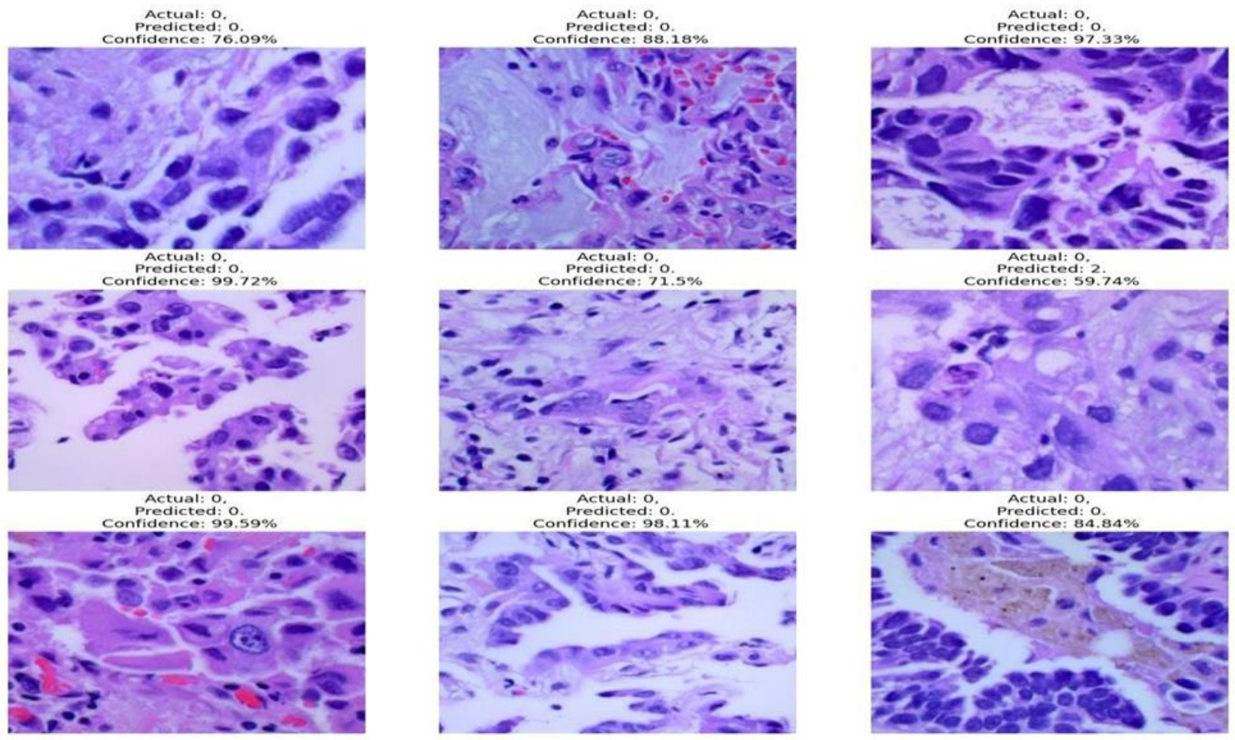
- `print("Predicted label:", class_names [predicted_label])`: This line prints the predicted class label.

- `break`: This line breaks out of the loop after processing the first image.



```python
import tensorflow as tf

plt.figure(figsize=(15, 15))
for images, labels in test_data:
    for i in range(9):
        ax = plt.subplot(3, 3, i + 1)
        plt.imshow(images[i])

        predicted_class, confidence = predict(model, images[i])

        # Convert labels[i] to integer using np.argmax()
        actual_class = class_names[np.argmax(labels[i])]

        plt.title(f"Actual: {actual_class},\n Predicted: {predicted_class}.\n Confidence: {confidence}%")

        plt.axis("off")
    break
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 138ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 52ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 50ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 63ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 53ms/step
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 73ms/step
```



1. Import Libraries: The code begins by importing the necessary libraries: `tensorflow` for working with neural networks and `matplotlib.pyplot` for visualization.

2. Set up Visualization: `plt.figure(figsize=(15, 15))` creates a figure window for displaying the results with a specified size.

3. Iterate Through Test Data: The code then iterates through the test dataset using a `for` loop. Each iteration processes a pair of `images` and corresponding `labels` from the `test_data`.

4. Display Image: Inside the loop, `plt.subplot(3, 3, i + 1)` divides the figure into a 3x3 grid and selects a specific subplot for each image. `plt.imshow(images[i])` displays the current image.

5. Predict Class: The `predict(model, images[i])` function uses the trained model to predict the class of the current image and returns the predicted class and its confidence score.

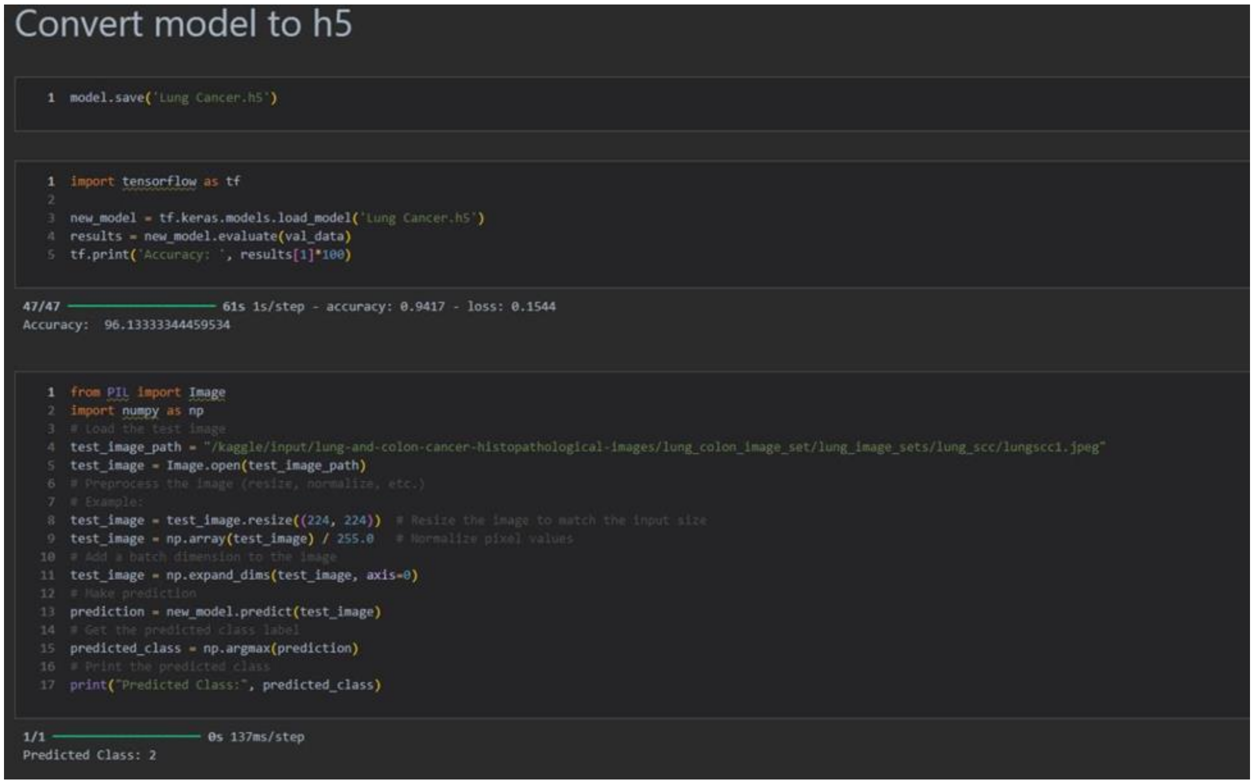
6. Get Actual Class: The `actual_class = class_names[np.argmax(labels[i])]` line converts the label (which might be a one-hot encoded vector) to the corresponding class name from the `class_names` list.

7. Set Plot Title: `plt.title(f"Actual: {actual_class}, \n Predicted: {predicted_class}.\n Confidence:

{confidence}%")` sets the title for the subplot, showing the actual class, predicted class, and the confidence level of the prediction.

8. Turn Off Axes: `plt.axis("off")` removes the axes labels and tick marks from the subplot, making the visualization cleaner.

9. Show Plots: Although not explicitly shown, the code would likely call `plt.show()` after the loop to display all the plots.



Convert model to h5

```
model.save('Lung Cancer.h5')
```

```
import tensorflow as tf

new_model = tf.keras.models.load_model('Lung Cancer.h5')
results = new_model.evaluate(val_data)
tf.print('Accuracy: ', results[1]*100)
```

```
47/47 ━━━━━━━━━━━━━━━━━━━━ 61s 1s/step - accuracy: 0.9417 - loss: 0.1544
Accuracy:  96.13333344459534
```

```
from PIL import Image
import numpy as np
# Load the test image
test_image_path = "/kaggle/input/lung-and-colon-cancer-histopathological-images/lung_colon_image_set/lung_image_sets/lung_scc/lungscc1.jpeg"
test_image = Image.open(test_image_path)
# Preprocess the image (resize, normalize, etc.)
# Example:
test_image = test_image.resize((224, 224))  # Resize the image to match the input size
test_image = np.array(test_image) / 255.0   # Normalize pixel values
# Add a batch dimension to the image
test_image = np.expand_dims(test_image, axis=0)
# Make prediction
prediction = new_model.predict(test_image)
# Get the predicted class label
predicted_class = np.argmax(prediction)
# Print the predicted class
print("Predicted Class:", predicted_class)
```

```
1/1 ━━━━━━━━━━━━━━━━━━━━ 0s 137ms/step
Predicted Class: 2
```

1. Loading the Model:

- `import tensorflow as tf`: Imports the TensorFlow library, which provides the tools for working with deep learning models.
- `new_model = tf.keras.models.load_model('Lung Cancer.h5')`: Loads the previously saved Keras model from the file 'Lung Cancer.h5'. This model presumably was trained to classify lung cancer images.

2. Evaluating the Model (Optional):

- `results = new_model.evaluate(val_data)`: This line evaluates the loaded model's performance on a set of validation data (`val_data`). This is optional but useful to assess how well the model generalizes to new data. The results would likely include metrics like accuracy and loss.
- `tf.print('Accuracy:', results[1]*100)`: Prints the model's accuracy on the validation data as a percentage.

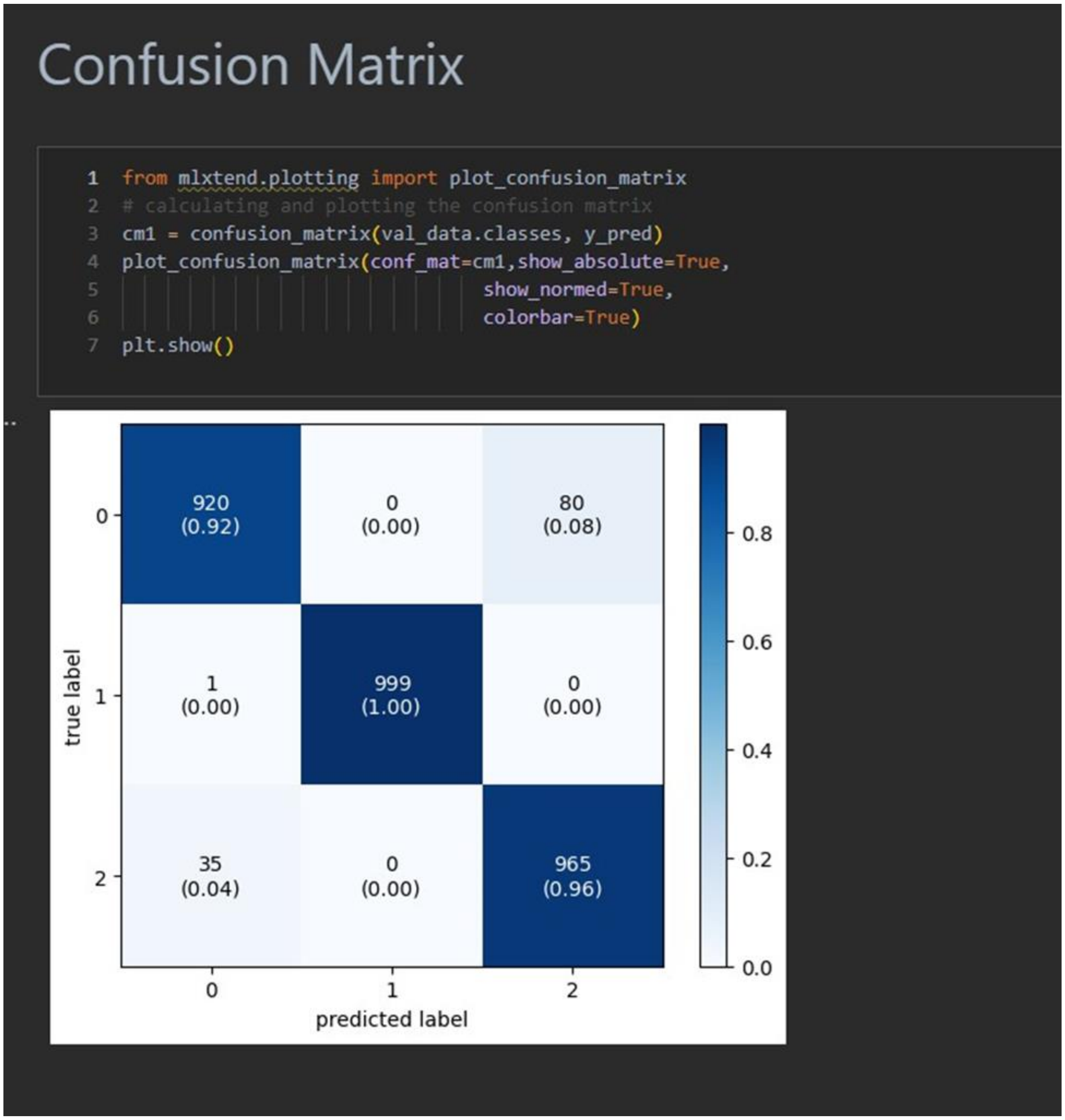
3. Predicting a New Image:

- Import Libraries:
- `from PIL import Image`: Imports the Python Imaging Library (PIL) for working with images.
- `import numpy as np`: Imports NumPy for numerical operations and array manipulation.
- Load and Preprocess Test Image:
- `test_image_path = "/kaggle/input/lung-and-colon-cancer-histopathological-

images/lung_colon_image_set/lung_image_sets/lung_scc/lungscc1.jpeg"`: Specifies the path to the image you want to classify.
- `test_image = Image.open(test_image_path)`: Opens the image using PIL.
- `test_image = test_image.resize((224, 224))`: Resizes the image to match the input size expected by your trained model.
- `test_image = np.array(test_image) / 255.0`: Converts the image to a NumPy array and normalizes pixel values to be between 0 and 1, a common practice for image processing in deep learning.
- `test_image = np.expand_dims(test_image, axis=0)`: Adds a batch dimension to the image. Many machine learning models expect data in a batch format, even if you're only predicting on a single image.
- Making the Prediction:
- `prediction = new_model.predict(test_image)`: Uses the loaded model to make a prediction on the preprocessed image.
- Interpreting the Prediction:

- `predicted_class = np.argmax(prediction)`: Finds the index of the highest probability output from the model. This index represents the predicted class (e.g., 0 for 'healthy', 1 for 'cancerous').

- `print("Predicted Class:", predicted_class)`: Prints the predicted class label based on the model's output.



Confusion Matrix

```python
from mlxtend.plotting import plot_confusion_matrix
# calculating and plotting the confusion matrix
cm1 = confusion_matrix(val_data.classes, y_pred)
plot_confusion_matrix(conf_mat=cm1,show_absolute=True,
                                show_normed=True,
                                colorbar=True)
plt.show()
```

1. Import:

- `from mlxtend.plotting import plot_confusion_matrix`: Imports the `plot_confusion_matrix` function from the `mlxtend.plotting` library. This function is designed to create visually appealing confusion matrix plots.

2. Calculate Confusion Matrix:

- `cm1 = confusion_matrix(val_data.classes, y_pred)`:
- This line calculates the confusion matrix.
- `val_data.classes`: This likely represents the true labels of the data used for evaluation (the "ground truth").
- `y_pred`: This likely represents the predictions made by your classification model on the same evaluation data.
- The resulting `cm1` is a matrix that summarizes how well your model's predictions align with the true labels.

3. Visualize Confusion Matrix:

- `plot_confusion_matrix(conf_mat=cm1, ...)`: This line calls the `plot_confusion_matrix` function to create the visualization. Here's what the arguments do:
- `conf_mat=cm1`: Specifies the confusion matrix (`cm1`) to be plotted.
- `show_absolute=True`: Displays the absolute number of instances in each cell of the matrix.

- `show_normed=True`: Displays the normalized values (proportions) within each cell.
- `colorbar=True`: Adds a color bar to the plot for easier interpretation of the values.

4. Display the Plot:

- `plt.show()`: This line displays the created confusion matrix plot.

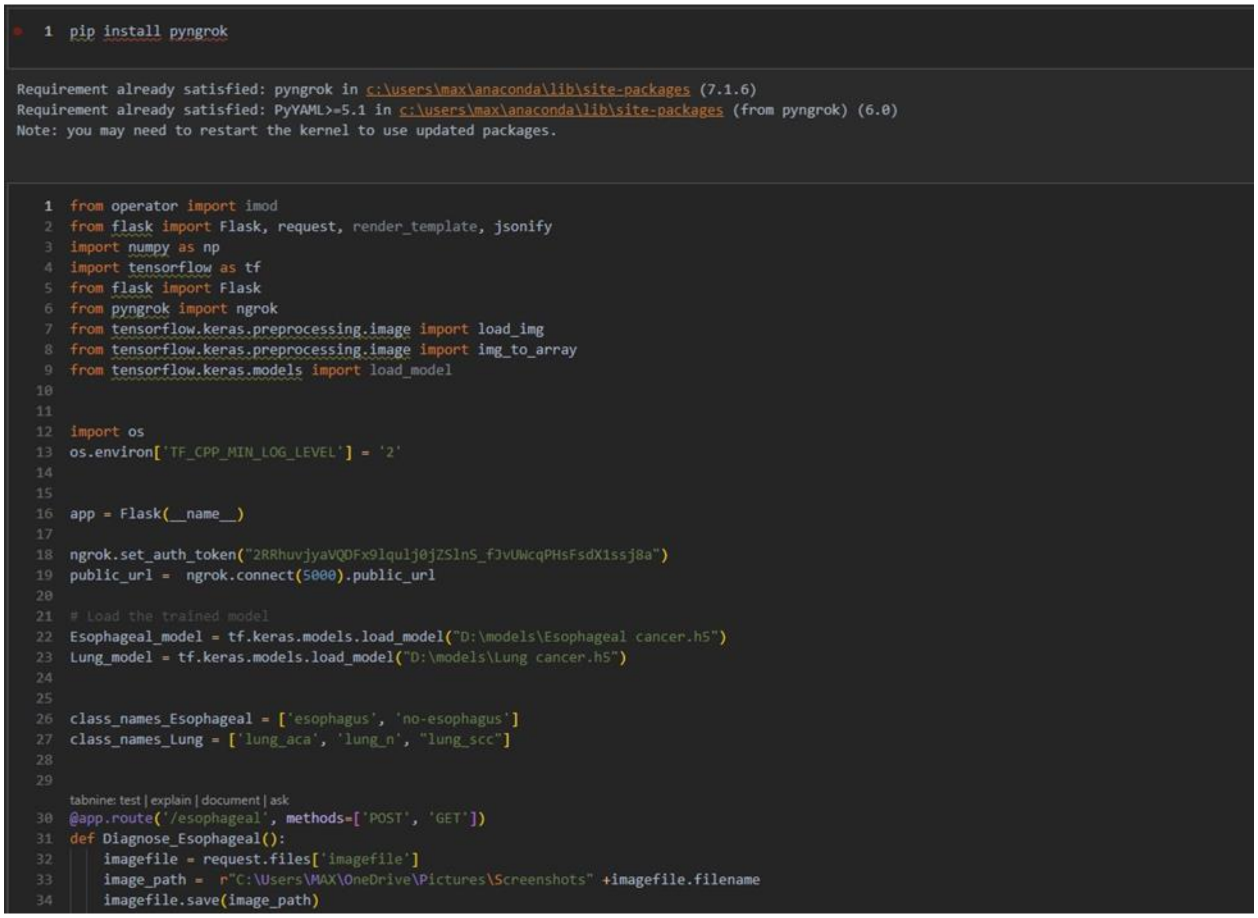
Understanding the Confusion Matrix:

A confusion matrix is a table that summarizes the performance of a classification model. It shows how many instances were correctly classified (diagonal elements) and incorrectly classified (off-diagonal elements).

Interpretation:

The numbers in the matrix represent counts of predictions. Here's an example interpretation of the given matrix:

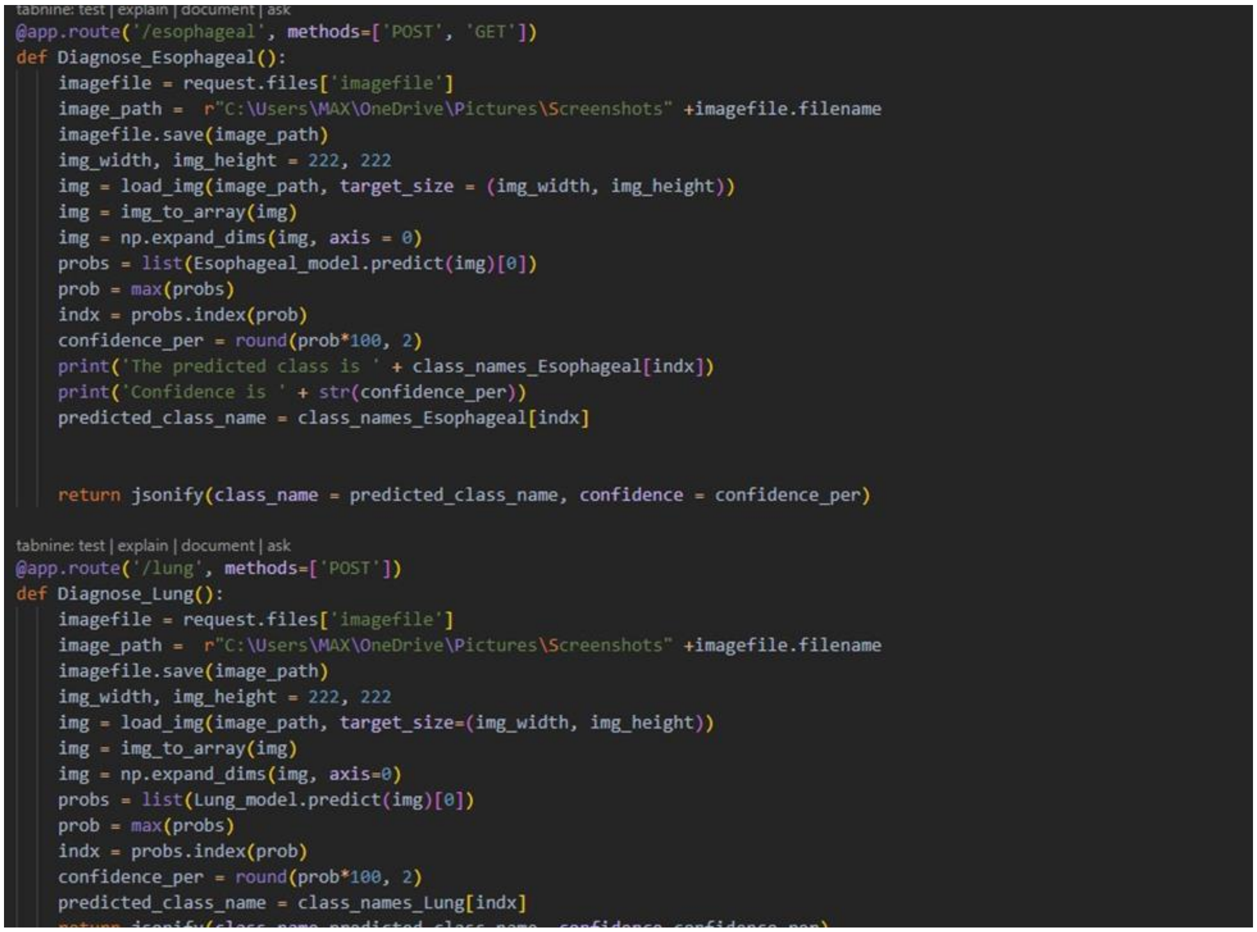
- True 0, Predicted 0 (920): The model correctly classified 920 instances as class 0.
- True 1, Predicted 1 (999): The model correctly classified 999 instances as class 1.
- True 2, Predicted 2 (965): The model correctly classified 965 instances as class 2.
- True 0, Predicted 1 (0): The model incorrectly classified 0 instances of class 0 as class 1.
- True 1, Predicted 0 (1): The model incorrectly classified 1 instance of class 1 as class 0.
- True 0, Predicted 2 (80): The model incorrectly classified 80 instances of class 0 as class 2.



```
pip install pyngrok
```

```
Requirement already satisfied: pyngrok in c:\users\max\anaconda\lib\site-packages (7.1.6)
Requirement already satisfied: PyYAML>=5.1 in c:\users\max\anaconda\lib\site-packages (from pyngrok) (6.0)
Note: you may need to restart the kernel to use updated packages.
```

```python
from operator import imod
from flask import Flask, request, render_template, jsonify
import numpy as np
import tensorflow as tf
from flask import Flask
from pyngrok import ngrok
from tensorflow.keras.preprocessing.image import load_img
from tensorflow.keras.preprocessing.image import img_to_array
from tensorflow.keras.models import load_model


import os
os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'


app = Flask(__name__)

ngrok.set_auth_token("2RRhuvjyaVQDFx9lqulj0jZSlnS_fJvUWcqPHsFsdX1ssj8a")
public_url =  ngrok.connect(5000).public_url

# Load the trained model
Esophageal_model = tf.keras.models.load_model("D:\models\Esophageal cancer.h5")
Lung_model = tf.keras.models.load_model("D:\models\Lung cancer.h5")


class_names_Esophageal = ['esophagus', 'no-esophagus']
class_names_Lung = ['lung_aca', 'lung_n', "lung_scc"]


tabnine: test | explain | document | ask
@app.route('/esophageal', methods=['POST', 'GET'])
def Diagnose_Esophageal():
    imagefile = request.files['imagefile']
    image_path =  r"C:\Users\MAX\OneDrive\Pictures\Screenshots" +imagefile.filename
    imagefile.save(image_path)
```



```
tabnine: test | explain | document | ask
@app.route('/esophageal', methods=['POST', 'GET'])
def Diagnose_Esophageal():
    imagefile = request.files['imagefile']
    image_path =  r"C:\Users\MAX\OneDrive\Pictures\Screenshots" +imagefile.filename
    imagefile.save(image_path)
    img_width, img_height = 222, 222
    img = load_img(image_path, target_size = (img_width, img_height))
    img = img_to_array(img)
    img = np.expand_dims(img, axis = 0)
    probs = list(Esophageal_model.predict(img)[0])
    prob = max(probs)
    indx = probs.index(prob)
    confidence_per = round(prob*100, 2)
    print('The predicted class is ' + class_names_Esophageal[indx])
    print('Confidence is ' + str(confidence_per))
    predicted_class_name = class_names_Esophageal[indx]


    return jsonify(class_name = predicted_class_name, confidence = confidence_per)

tabnine: test | explain | document | ask
@app.route('/lung', methods=['POST'])
def Diagnose_Lung():
    imagefile = request.files['imagefile']
    image_path =  r"C:\Users\MAX\OneDrive\Pictures\Screenshots" +imagefile.filename
    imagefile.save(image_path)
    img_width, img_height = 222, 222
    img = load_img(image_path, target_size=(img_width, img_height))
    img = img_to_array(img)
    img = np.expand_dims(img, axis=0)
    probs = list(Lung_model.predict(img)[0])
    prob = max(probs)
    indx = probs.index(prob)
    confidence_per = round(prob*100, 2)
    predicted_class_name = class_names_Lung[indx]
    return jsonify(class_name=predicted_class_name, confidence=confidence_per)
```

The code is a simple web application using Flask and ngrok that diagnoses esophageal and lung cancer using pre-trained deep learning models built with TensorFlow and Keras. Below is a detailed explanation of the code:

1. 1.Importing Libraries:

1.1. These imports bring in necessary libraries like Flask for creating the web server, TensorFlow for loading and using deep learning models, and pyngrok for exposing the local server to the

internet.

2. 2.Setting Up Flask and ngrok:

2.1. Here, we set up the Flask environment and configure ngrok with an authentication token to create a public URL for our local server.

3. 3.Loading Pre-trained Models:

3.1. We load the pre-trained models for esophageal and lung cancer diagnosis.

4. 4.Defining Class Names:

4.1. We define the class names that the models can predict.

5. 5.Defining Application Routes:

5.1. .Route for diagnosing esophageal cancer:

5.2. In this route, we receive an image file from the user, save it, process it, and feed it to the esophageal cancer model. Then, we return the result (class name and confidence).

5.3. .Route for diagnosing lung cancer:

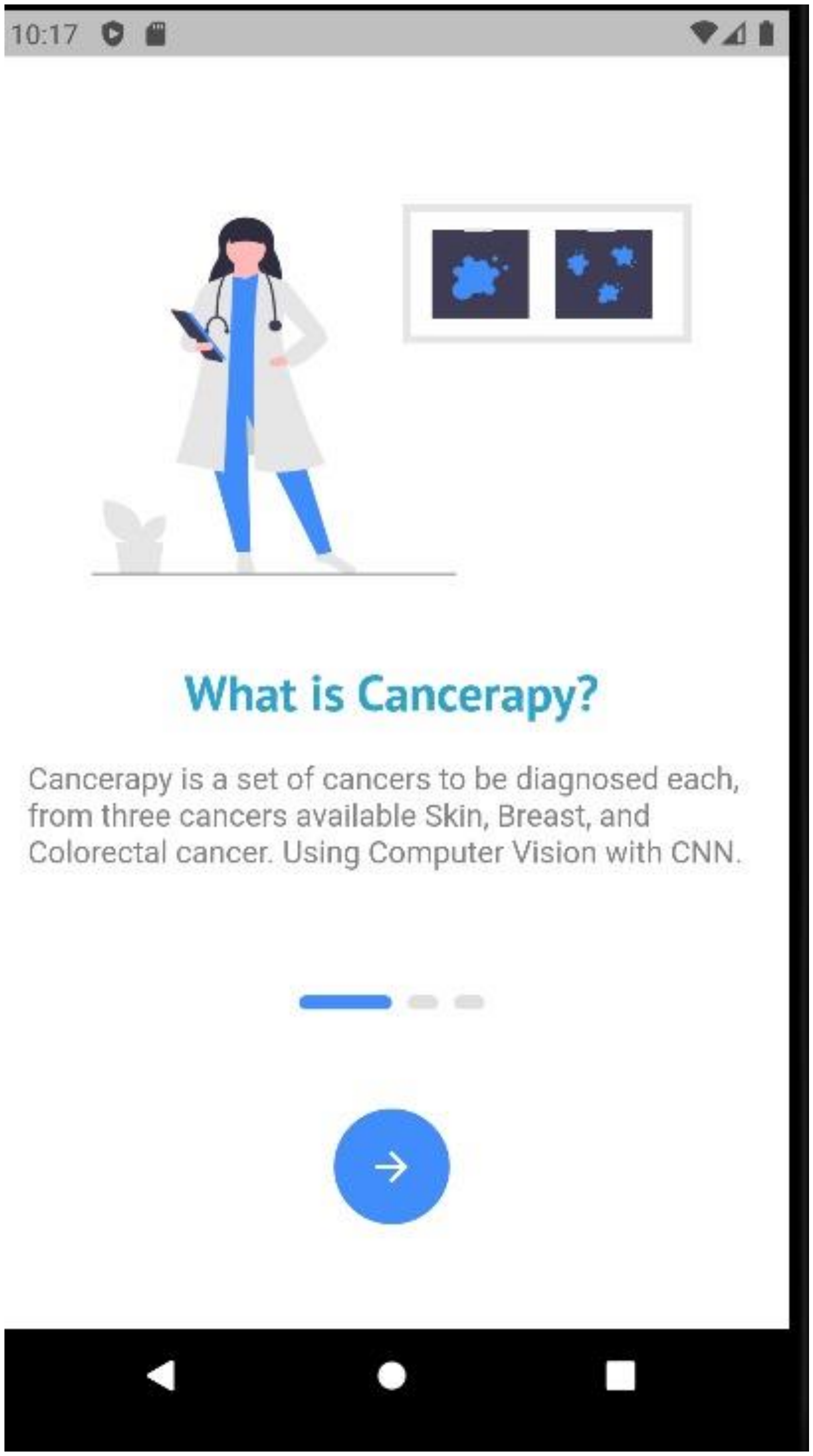
6. The same process as in the esophageal route, but using the lung cancer model.

7. 6.Running the Application:

7.1. Finally, we print the ngrok public URL to access the application from anywhere and start the Flask server on port 5000.

CHAPTER FIVE

Prototype

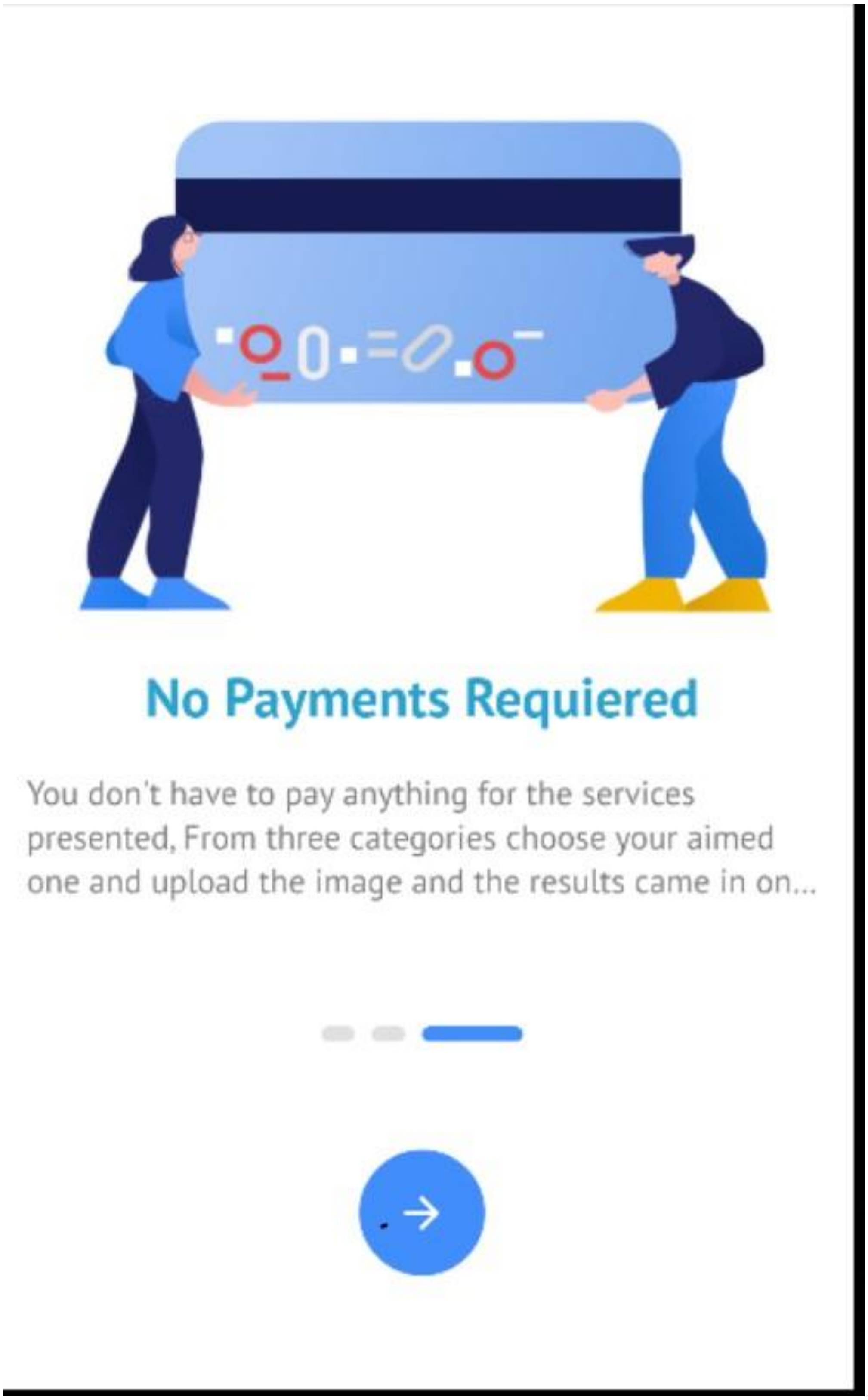


What is Cancerapy?

Cancerapy is a set of cancers to be diagnosed each, from three cancers available Skin, Breast, and Colorectal cancer. Using Computer Vision with CNN.

The image shows a mobile app screen explaining what Cancerapy is. The app uses computer vision to diagnose cancers, specifically skin, breast, and colorectal cancer. There's a doctor in the image, suggesting the app is meant for medical professionals.

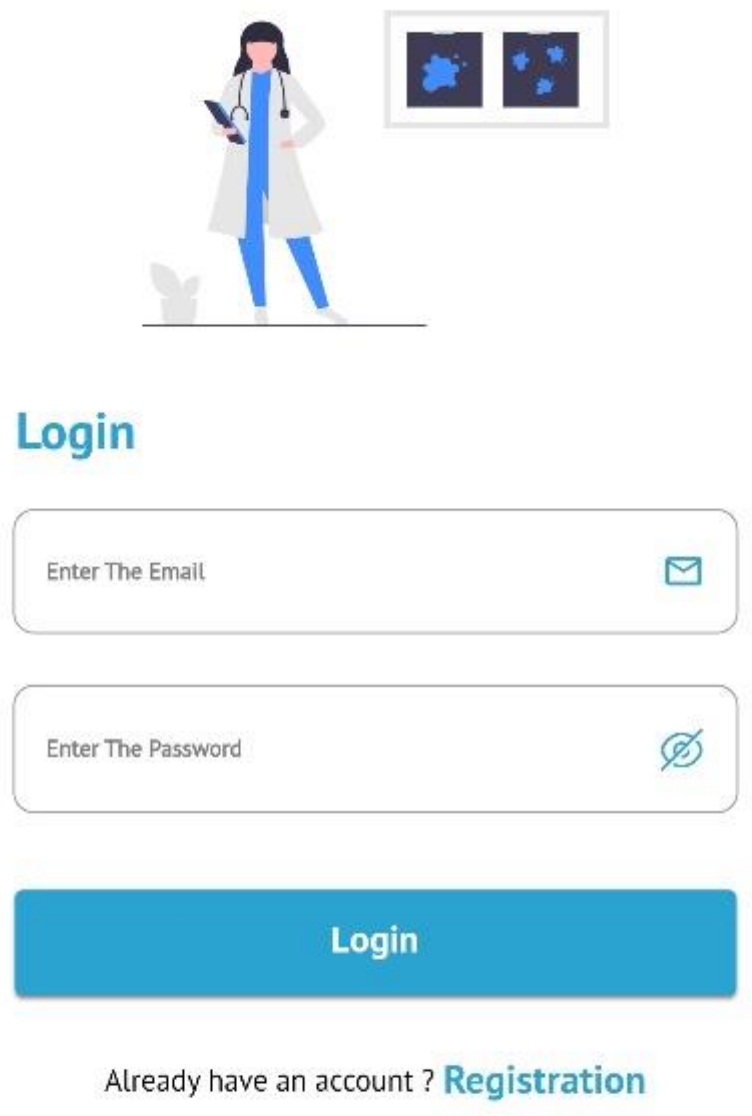
The design is clean and simple, with a blue theme. The app seems to be user-friendly and easy to navigate.



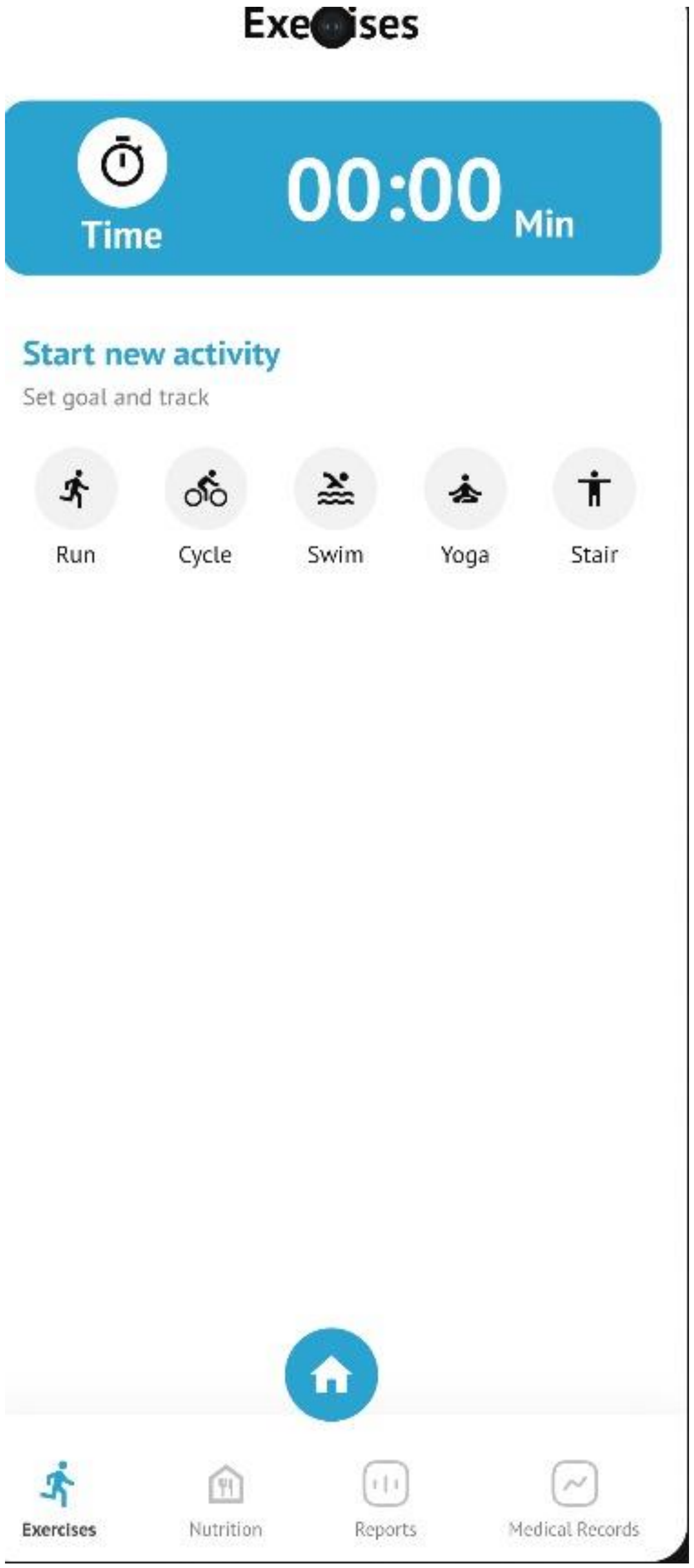
No Payments Required

You don't have to pay anything for the services presented, From three categories choose your aimed one and upload the image and the results came in on...

The image shows two cartoon people carrying a credit card with the words "No Payments Required" written below it. This suggests that the service being advertised is free of charge. The text below explains that users can choose from three categories and upload an image to receive results. The arrow at the bottom likely indicates a button to continue or proceed with the service. Overall, the image promotes a free service that uses image uploads to generate results.

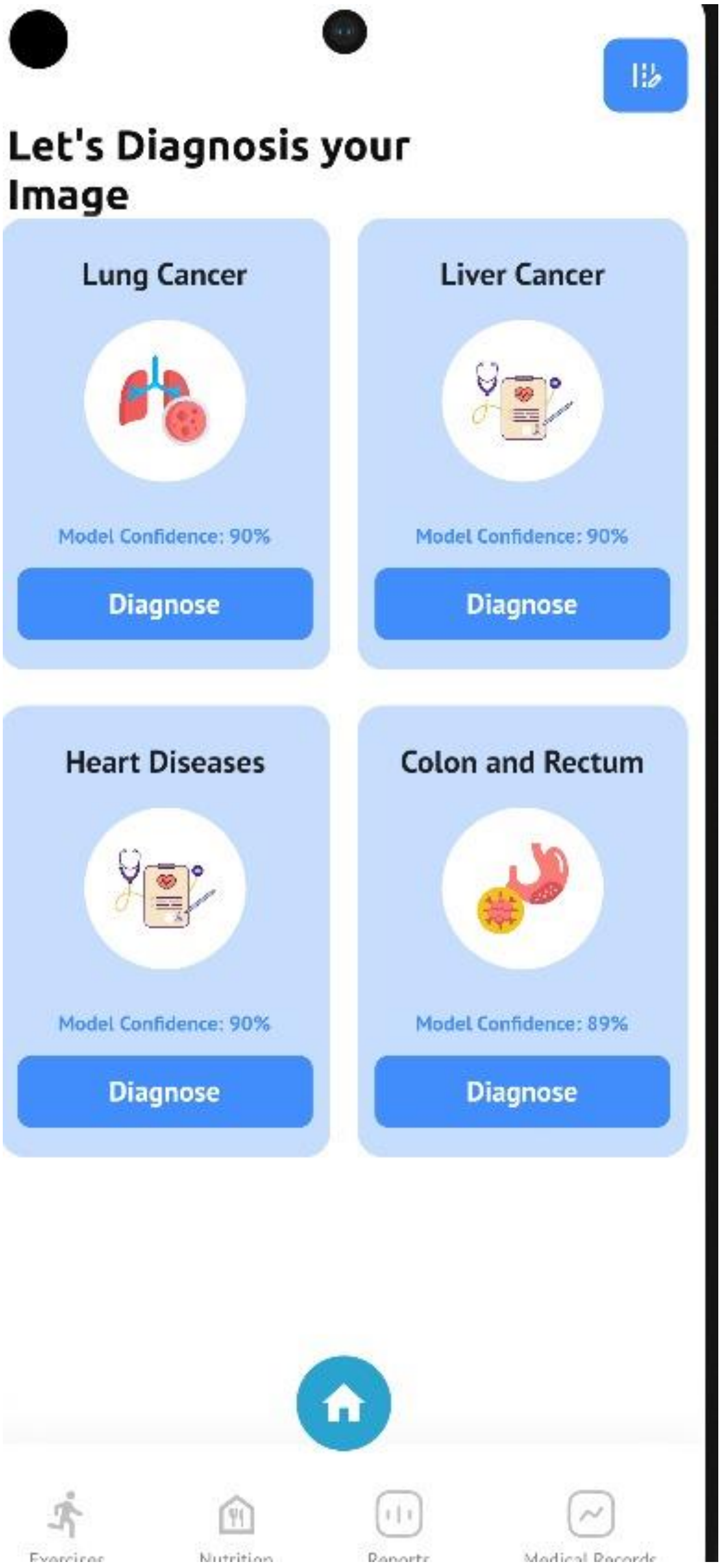


This image is a login page for a website. It has a title "Login" and two fields: "Enter The Email" and "Enter The Password." Below the fields is a button labeled "Login." Below the button, there is a message saying "Already have an account? Registration" with a link to registration. The image also features a cartoon illustration of a female doctor standing with a tablet in her hand, with two x-ray images in the background. This design suggests that the website is related to healthcare. The design is simple and user-friendly, which is typical of login pages.


Exe ises
Time
00:00 Min
Start new activity
Set goal and track
Run
Cycle
Swim
Yoga
Stair
Exercises
Nutrition
Reports
Medical Records

This is a screenshot of an app that tracks your exercise activity.

It shows the current time as 00:00 minutes and asks the user to start a new activity by selecting one of the five activities: run, cycle, swim, yoga, or stair.



Let's Diagnosis your Image

Lung Cancer

Model Confidence: 90%

Diagnose

Liver Cancer

Model Confidence: 90%

Diagnose

Heart Diseases

Model Confidence: 90%

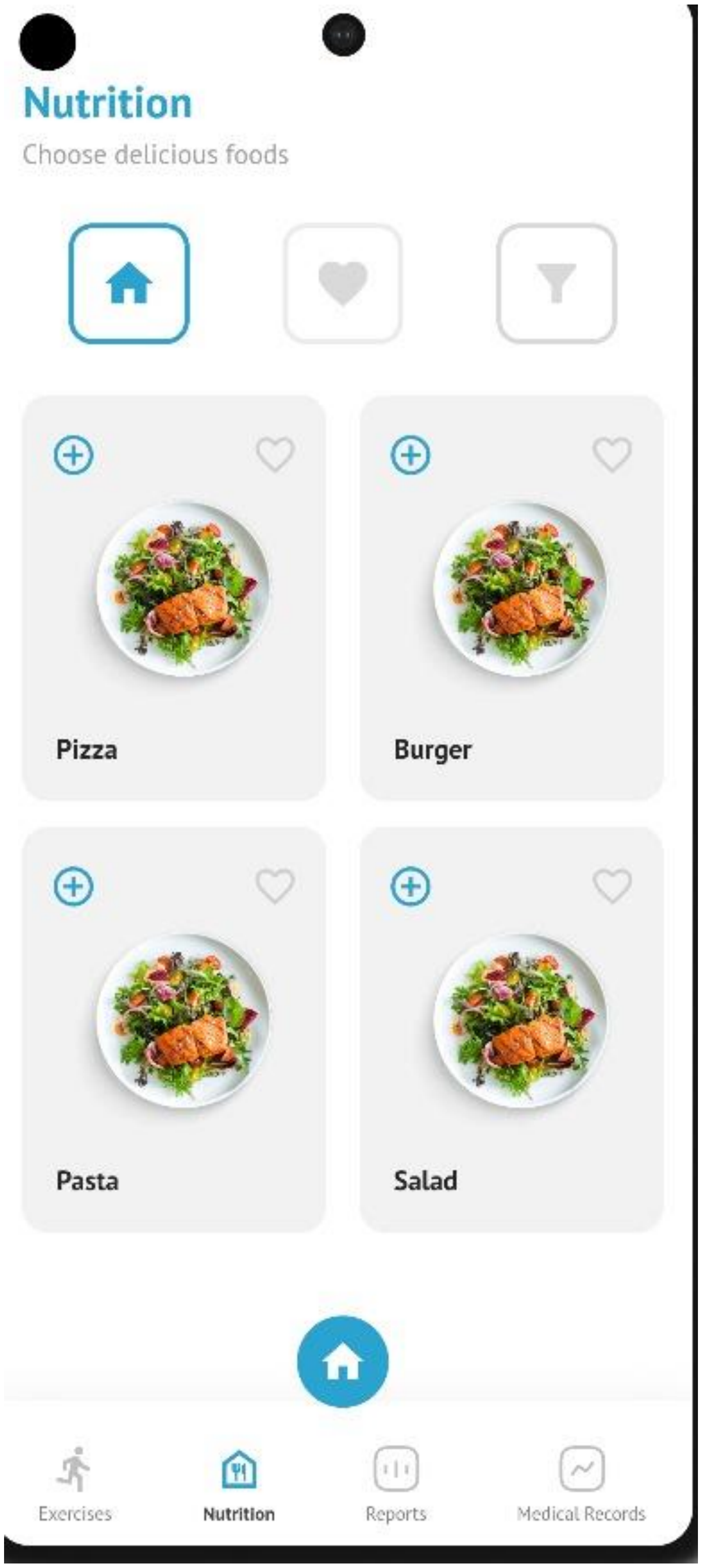
Diagnose

Colon and Rectum

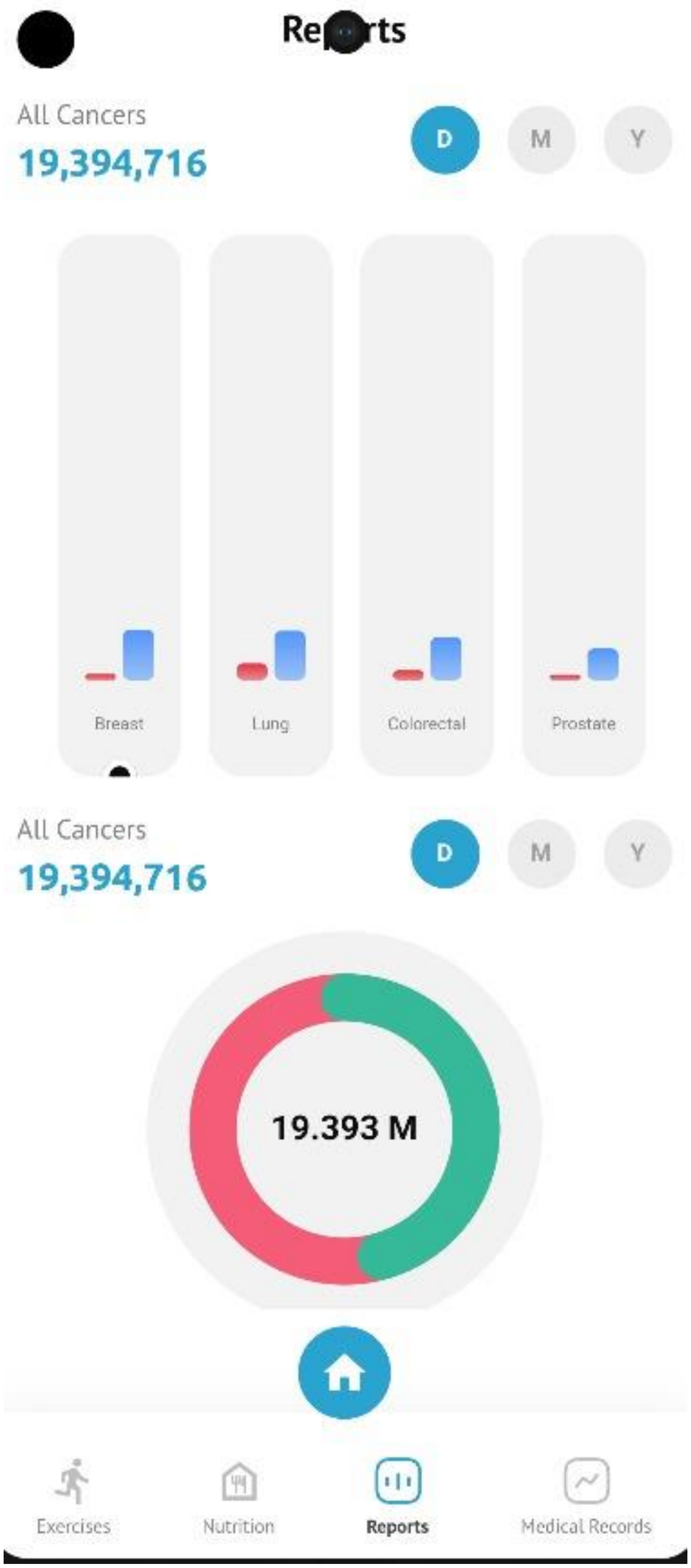
Model Confidence: 89%

Diagnose

The image is a mockup of a medical app that helps users diagnose diseases based on their symptoms. The app has a home screen with four buttons that represent different diseases: lung cancer, liver cancer, heart disease, and colon and rectum cancer. Each button is accompanied by an icon and a confidence score. The user can click on a button to learn more about the disease and its symptoms. There are also buttons at the bottom for exercises, nutrition, reports, and medical records.


Nutrition
Choose delicious foods
Pizza
Burger
Pasta
Salad
Exercises
Nutrition
Reports
Medical Records

This is a screenshot of a mobile app for tracking nutrition. The app allows users to choose from a variety of foods, including pizza, burger, pasta, and salad. Users can add these foods to their diet by tapping the plus icon next to them. The bottom of the screen shows options to navigate to other features in the app, including Exercises, Nutrition, Reports, and Medical Records. The home icon in the center is to return to the main screen.

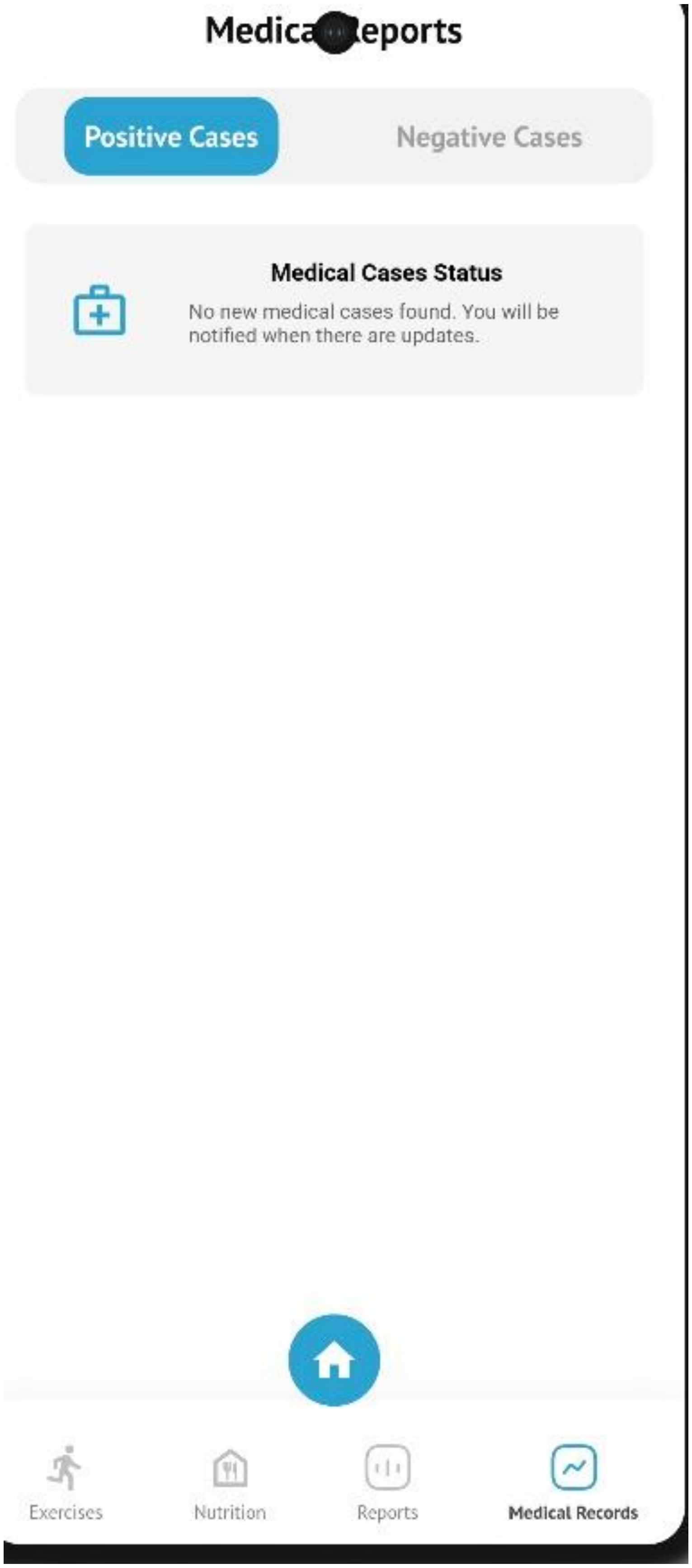

Reports
All Cancers
19,394,716
D
M
Y
Breast
Lung
Colorectal
Prostate
All Cancers
19,394,716
D
M
Y
19.393 M
Exercises
Nutrition
Reports
Medical Records

The image shows a dashboard with information about cancer rates. The top part of the dashboard displays the total number of cancer cases: 19,394,716. Below this, there are four bars representing different types of cancer: Breast, Lung, Colorectal, and Prostate. Each bar shows the number of cases for that type of cancer.

The lower part of the dashboard shows a pie chart with the total number of cancer cases represented as 19.393 Million. This chart is likely to be showing the percentage of each cancer type from the top part of the dashboard.

The bottom of the dashboard shows five icons for navigation: Exercises, Nutrition, Reports, Medical Records, and a home button. This suggests that this dashboard is part of a larger health and wellness app or website.

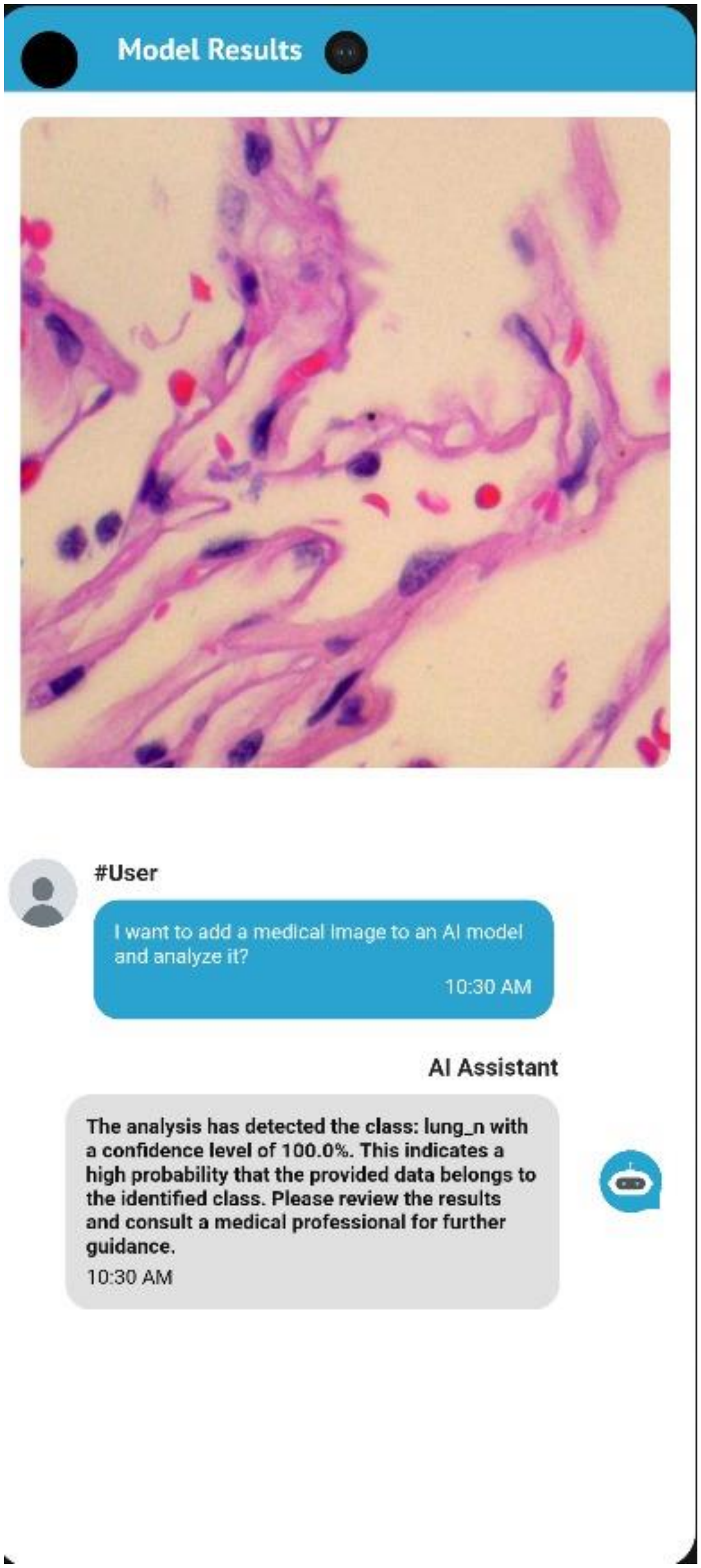
This dashboard provides a quick overview of cancer rates for different types of cancer. It could be used by healthcare professionals to track trends or by individuals to learn more about cancer statistics.

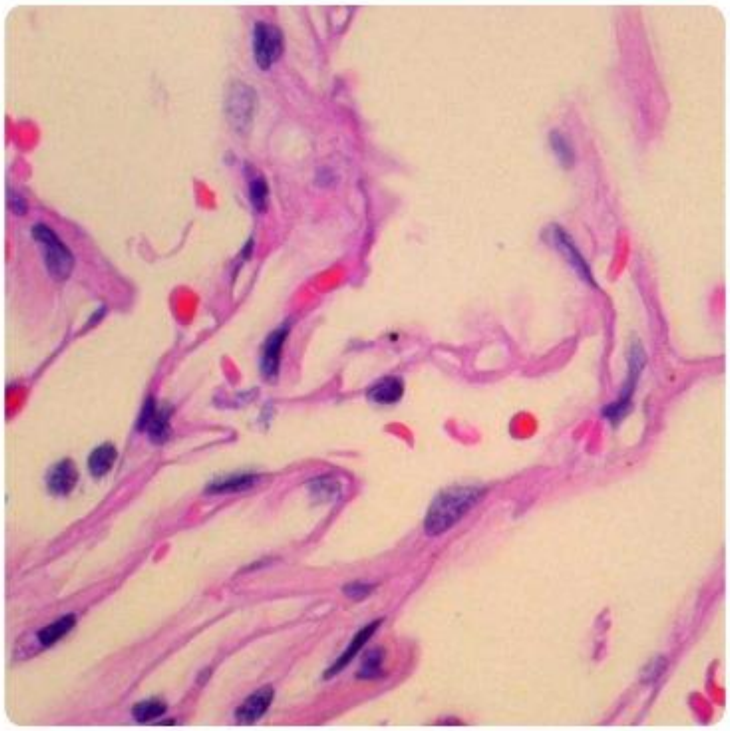


Medical Cases Status

No new medical cases found. You will be notified when there are updates.

The image shows a screen from a mobile app that tracks medical cases. The user can view either positive or negative cases, and the app shows a message that there are no new medical cases found. The image also shows a navigation bar at the bottom with options for Exercises, Nutrition, Reports, and Medical Records. This suggests the app aims to help users manage their health with various features.





#User

I want to add a medical image to an AI model and analyze it?

10:30 AM

AI Assistant

The analysis has detected the class: lung_n with a confidence level of 100.0%. This indicates a high probability that the provided data belongs to the identified class. Please review the results and consult a medical professional for further guidance.

10:30 AM

This image shows a conversation between a user and an AI assistant. The user wants to analyze a medical image with an AI model. The AI model has identified the image as "lung_n" with a 100% confidence level, meaning it's highly likely the image is of a lung. The AI assistant warns that this is a medical image and encourages the user to consult a medical professional for further guidance.