Al-Azhar University

Faculty of Engineering,

Systems & Computers Engineering Department

Project Title:

Strabismus Eye Detection Application

Submitted By

1- Asmaa Elboghdady Elwehady < 20248615 >
2- Asmaa Tharwt Ahmed < 20248616 >
3- Aisha Mahmoud Fathy < 20248717 >
4- Fatma Alzhraa Nagah Said < 20248729 >
5- Fatma Khaled Tayel < 20248732>
6- Heba Farid El Sherbiny < 20248777 >
7- Walaa Elsayed Mohamed < 20248783>

Supervised by

Dr. Marwa Selim

2024/2023

Acknowledgement

I would like to extend my heartfelt appreciation to the divine forces that have bestowed blessings upon us, enabling us to achieve our goals. It is with immense joy and a profound sense of fulfillment that I address the successful completion of our project. It is crucial to acknowledge the individuals who have played a significant role in making it possible.

I would like to express my sincere gratitude to our esteemed project supervisor, Dr. Marwa Refaey, for her exceptional guidance, unwavering encouragement, and invaluable suggestions throughout the project's journey. Her expertise and support have been instrumental in our pursuit of success.

Furthermore, I am immensely grateful to Al-Azhar University for providing us with a top-tier education and equipping us with the necessary resources to develop this project. The knowledge and skills acquired from this esteemed institution have been indispensable in our pursuit of excellence.

Lastly, I would like to extend my heartfelt thanks to all the individuals who generously contributed their talents and dedicated efforts to assist us in the development of this project. Their unwavering support and commitment have been pivotal in our journey towards accomplishment.

Abstract

Strabismus, commonly known as crossed eyes, is a vision condition where the eyes are misaligned and unable to focus on the same object simultaneously. Early detection and intervention are crucial for proper treatment and preventing vision complications. This project presents the development of a Strabismus Eye Detection System, a user-friendly application offering three key functionalities:

1. **Informational Resources**: Providing access to educational articles on strabismus, various eye diseases, and general eye care practices.
2. **Strabismus Detection**: Implementing a camera-based system that analyzes eye positions in captured images and estimates the degree of strabismus deviation (percentage). It is important to emphasize that this feature is for informational purposes only and should not replace a professional diagnosis by an ophthalmologist.
3. **Eye Care Exercises & Guidelines**: Offering users a dedicated section with recommended eye exercises and guidelines for maintaining healthy vision.

This project aims to contribute to improved awareness about strabismus and empower users with preliminary detection capabilities. However, a formal diagnosis from a qualified ophthalmologist remains crucial for proper treatment planning.

Table Of Content

LIST OF FIGURES

Chapter 1 :Introduction & Overall Description

1.1 The Triple Constraint of our project:

• Project Scope

• Time

• Cost

Project scope:

The Strabismus Eye Detection System project aims to develop a user-friendly application with a specific focus on three key functionalities:

Inclusions:

- **Informational Resources:**
  - Develop a database containing informative articles on:
    - Strabismus causes, symptoms, and types.
    - Common eye diseases and their signs.
    - General eye care practices for maintaining healthy vision.
- **Strabismus Detection (Informational Only):**
  - Implement a camera-based system that analyzes eye positions captured through the device's camera.
  - Develop an image processing algorithm to estimate the degree of strabismus deviation (percentage).
    - Note: Emphasize throughout the project that this feature is for informational purposes only and should not replace a professional diagnosis.
- **Eye Care Exercises & Guidelines:**
  - Integrate a section with recommended eye exercises for improving eye health and coordination.
  - Include guidelines for maintaining healthy vision habits (e.g., screen time limitations, proper lighting).

Exclusions:

- **Medical Diagnosis and Treatment:** This application is not intended to diagnose or treat any medical condition. Users are encouraged to seek professional medical advice from ophthalmologists for proper diagnosis and treatment plans.
- **Advanced Eye Tracking:** The system will focus on a basic estimation of strabismus deviation, not detailed eye movement tracking functionalities.
- **Data Collection and Storage (Optional):**
  - Decide if the application will collect user data (e.g., anonymized usage statistics).
  - If data collection is involved, define clear procedures for user consent, data anonymization, and secure storage, adhering to relevant data privacy regulations.

Deliverable:

- A fully functional Strabismus Eye Detection System application.
- Project documentation outlining the development process, functionalities, and limitations.

Success Criteria:

- A user-friendly and informative application that effectively delivers educational content on strabismus and eye care.
- A functional strabismus detection feature (with clear disclaimers) that provides a preliminary estimation of eye misalignment.
- A well-structured and documented project demonstrating your technical skills and understanding of strabismus.

1.2 Work elements and techniques:

- **Mobile Application**: The main work involves developing an Android mobile application to estimate strabismus, provide information about eye diseases, and help users maintain eye health by providing exercises.
- **Machine Learning Model**: Using deep learning (TensorFlow library) to detect strabismus.
- **Techniques**: Object detection API using TensorFlow for image detection.

1.2.1 Mobile application

The main work of the android mobile application which estimate strabismus, provide information about eye's diseases and help user to keep its health by providing exercises.

1.2.2 Machine learning model:

In our tracking system we need a machine algorithm meet which our needs and be compatible with our data so we use deep learning by using TensorFlow library to detect the error.

1.2.3 Techniques

Object detection API using TensorFlow (Image detection):

it takes the object like the captured picture and make detection on it to detect if this user suffer from any eye's diseases or it is normal and return the result .

1.3 USED TOOLS

· **Designing Interface (UI)**: Using Android, HTML, XML to design user interfaces.

· **Local Database**: Using RoomDB to store articles and instructions locally for offline access.

· **Deep Learning**: Using TensorFlow for estimation of abnormal eyes.

· **Programming Languages**: Using various languages such as Python for the model, and Kotlin and Android programming languages for app development.

1.3.1 Design user interfaces (UI)

The goal of user interface design is to produce a user interface which makes

it easy, efficient, and enjoyable to operate a machine in the way which

produces the desired result.

In our project we will use android, HTML, XML

1.3.2 Local Database

To ensure efficient storage and retrieval of this informative content, the application leverages Room, a popular Android persistence library based on SQLite. Room simplifies database interactions, allowing us to store articles, instructions, and exercises locally in a structured and easily accessible manner for users. This approach enhances the application's user experience by providing offline access to valuable information and resources, even when an internet connection is unavailable.

1.3.3 Deep learning

Deep leaning is a machine learning approach depends on neural

networks that produce a very good accuracy and need huge

dateset to learn and find a pattern between data.

We use TensorFlow library to make estimation for abnormal eyes

1.3.4 Programming languages

A programming language is designed to communicate the user instructions

to a machine and can be used to create programs to control the behavior of a machine or to express algorithms.

We use more than one language in our project as:

1. The algorithms of machine learning will execute using android programming language.

2. The interaction between user interface and the system will also be controlled by android programming language.

3. Using RoomDB library to interact with database.

4. Using Python in model.

5- Using Kotlin in android.

Stakeholders for the Strabismus Eye Assistant App:

- **Project Team:** Develops and maintains the app, requiring clear goals, timelines, and resources.
- **Target Users:** Utilize the app to learn about strabismus, potentially check for misalignment (with disclaimers), and access eye care resources. They need a user-friendly interface, accurate information, and data privacy.
- **Secondary Stakeholders:**
    - **Ophthalmologists:** Benefit from the app raising awareness and encouraging users to seek professional help, requiring clear disclaimers within the app.
    - **Public Health Organizations:** Can collaborate on promoting eye health if the app's content aligns with their initiatives.
- **Tertiary Stakeholders:**
    - **Tech Reviewers & Media:** Increase app visibility through positive reviews and coverage, requiring a well-developed and informative application.
    - **Android Developer Community:** Provides knowledge and potential collaboration opportunities, with the need for adherence to best practices and potential contributions.

1.6 Project steps

- Study and analyze the project idea.
- Search and choose the best algorithm for the project.
- Model and simulate the system.
- Solve problems and errors in the system.
- Deploy and implement the system.
- Test the accuracy of the app and machine learning model.
- Evaluate the system for future improvements.

Chapter2: System Analysis

2.1 Introduction

Systems analysis is a problem-solving technique that decomposes a system ,into its component pieces for the purpose of the studying how well those ,component parts work and interact to accomplish their purpose.

It is a main step in any business as firstly we determine the requirements,

available resources, system parts and how each part interacts with another parts.

System analysis in information technology is widely used as we identify the system requirements, model the system and testing it to repair errors before final deployment the whole system.

2.2 System development life cycle

1. Planning

In this step we identify goals, establish strategies to achieve goals and

develop plans to integrate activities.

2. Analysis

At first, we should analysis the whole system requirements, available

resources, and interaction between the different parts in the system and we will use in our project the UML analysis model.

3. Implementation

Put the system in the action.

4. Review

Testing the system to ensure there is no errors and will work correctly after

Deployment.

2.3 System requirements

It represents what system need to be implemented and this term is widely used in software engineering and in analysis any systems. There is more than type in the system requirements as software requirements, functional requirements and non-functional requirements.

2.3.1 Software requirements

1. Machine learning to detect strabismus
2. Software to allow the system actors to interact with the system.
3. Android operating system.

2.3.2 Functional requirements

According to our project goal we have to do following things …

1. User take a photo to his eyes.
2. System converted it to PNG.
3. The model will detect it.

2.3.3 Non-Functional requirements

1. Security
2. Safety
3. Accuracy
4. quality
5. Speed
6. Availability
7. Usability

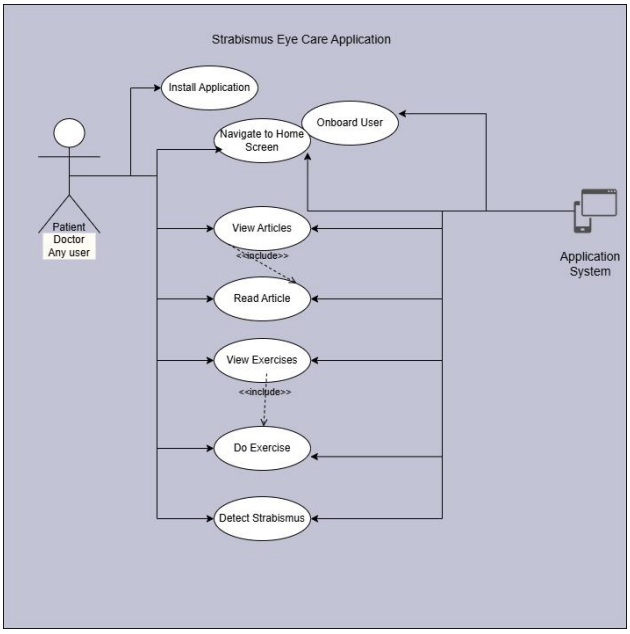
2.4 UML analysis model

It is referring to “Unified Modeling Language” that is a modeling language in the field of software engineering, which is designed to provide a standard way to visualize the design of a system.

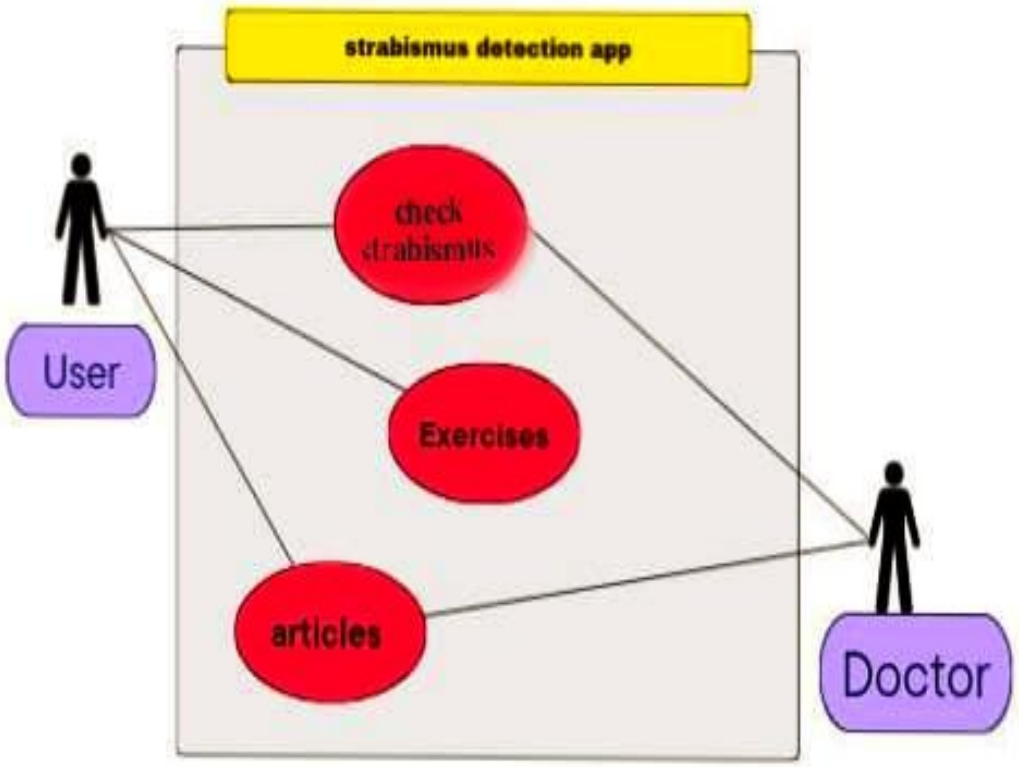
UML has many types of diagrams which are divided into two categories structured UML

diagrams and behavioral UML diagrams.

Block diagram of the system:



2.4.1 Use case Diagram



<< Fig 2.1 >>

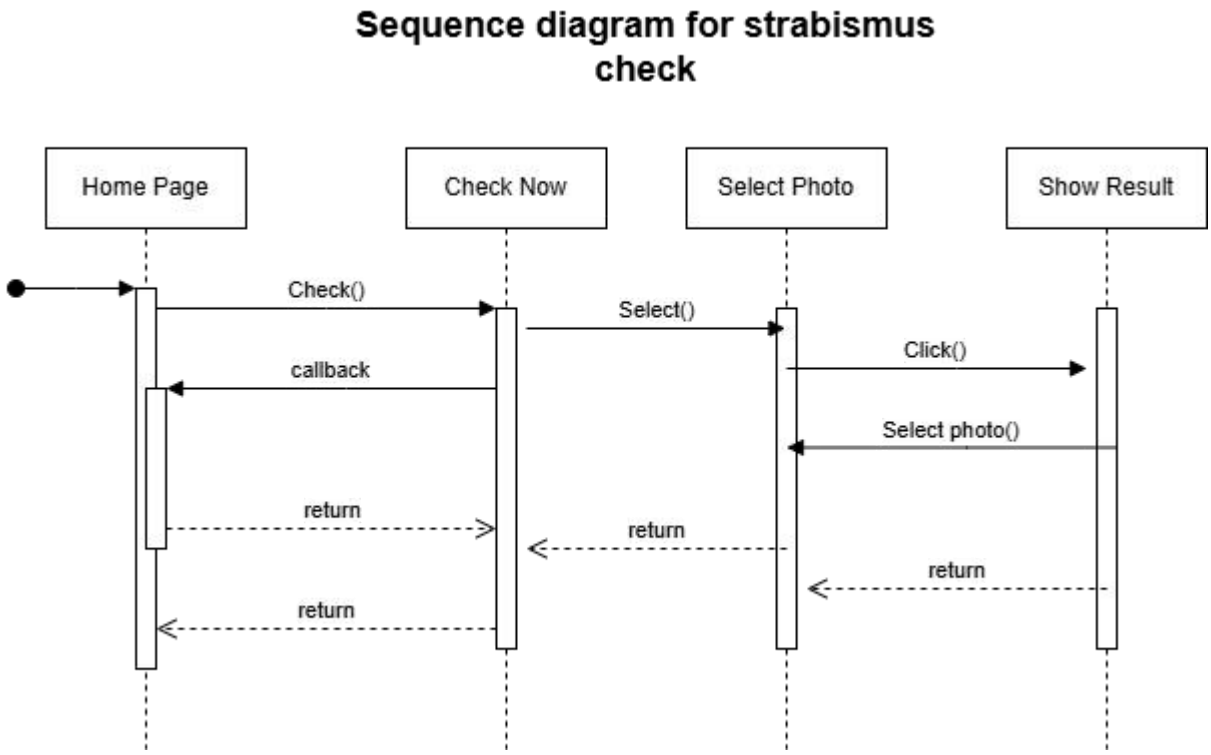
UML use case diagram: in our application actor can be user or doctor and there are three functions:

1) check strabismus

2) 2) Exercises

3) Articles

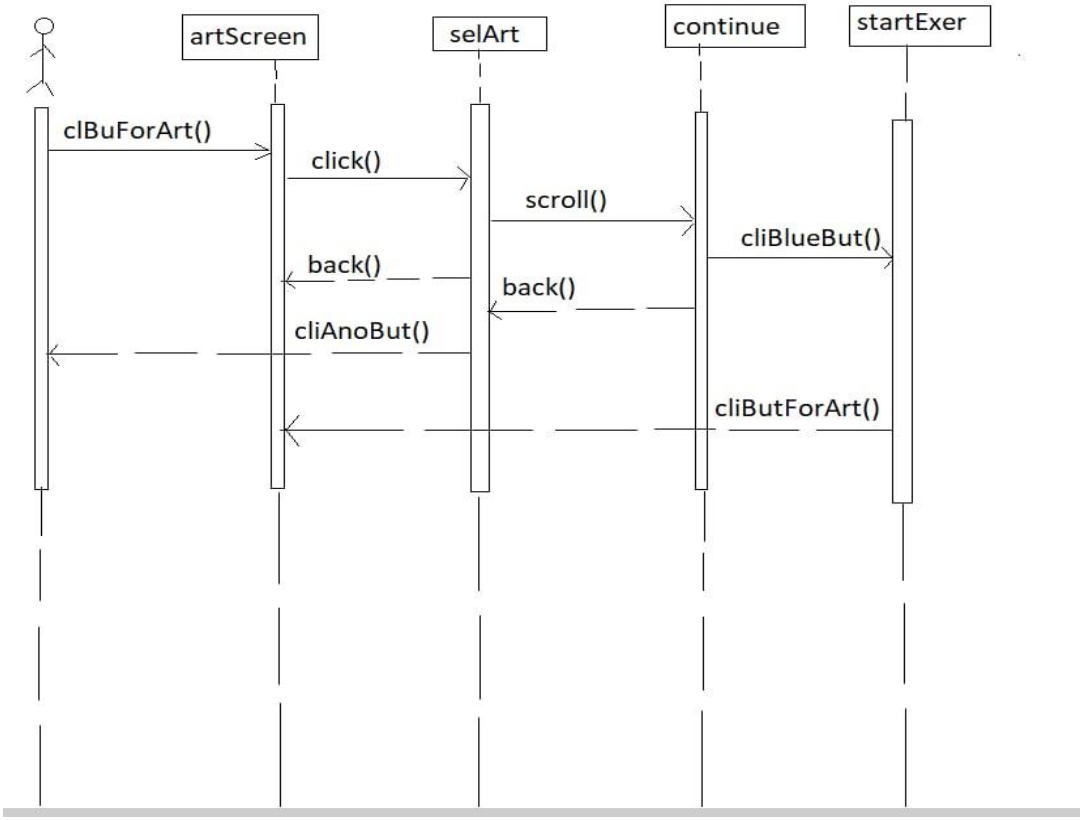
You can choose any function through button navigation.



<<Fig 2.2>>

To check strabismus you should do the following steps: click on check button, Select image from your storage and AI model check if the user have strabismus or not and you still be able to check any image and ready to do any author check.if user click another button, he/she will be go to another function.

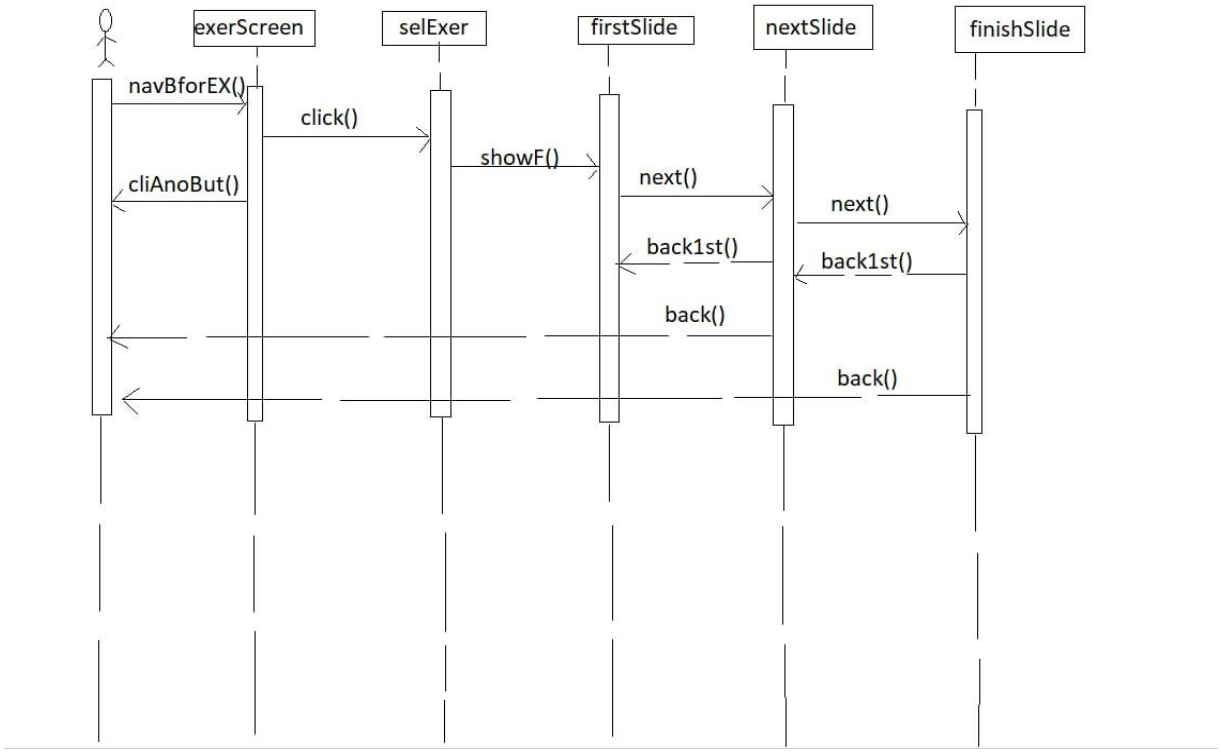
sequence diagram for articles:



<< Fig 2.3 >>

If the user click button navigation for articles the user will be on the article screen and can select particular article by click on it and the user can continue reading article by scrolling down and can start exercise by click on the blue button

Sequence digram for Exercises:



<< Fig 2.4 >>

If the user click button navigation for exercises he/she will be on exercise screen and have many exercises and to select exercise he/she must choose particular exercise by click on it and after doing this the user see the first slide of exercise and if the user click next he/she go to the next slide and so on and if he/she click back this action lead to go to the exercise screen.

Chapter 3: Project Design

3.1 Introduction

In the Strabismus detection app, the data is one of the most important components to build the application, it plays vital role in the structure of it. So, it had to be gathered and arranged in an orderly manner, management and effectiveness of running the data is essential to transfer and apply processes on it. So, it is important to build accurate database in order to deal easily and accurately to obtain results. In the Strabismus detection app,we deal with static data when user need to Know more information about anything is related to eyes through articles or exercises.

3.2 Used Data

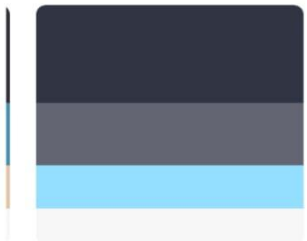
The application mainly based on the data that are stored in our database system:

- Articles which should be displayed when user need to read.
- Exercise and guidelines which user need to do it.

3.3 UI/UX Design

Our pallet

We chose this pallet as it’s one of the most suitable pallets for medical field.



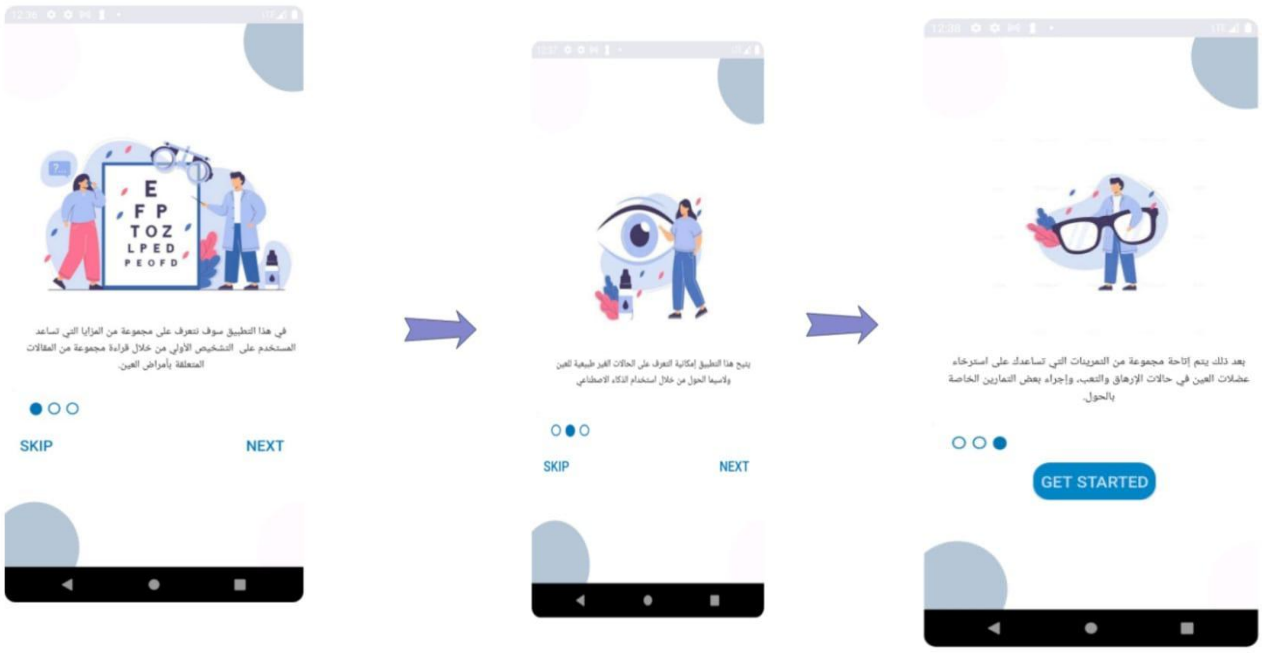
3.3.1 Mobile App screens

All designs are XML files.

- This the first screen which will face the user when he opens the app

Figure 2.3 start strabismus app

When the user download and install app for the first time ,it will guide him to know the goal of app and its features through welcome screens , There is 3 screens which explain the 3 functions of app.

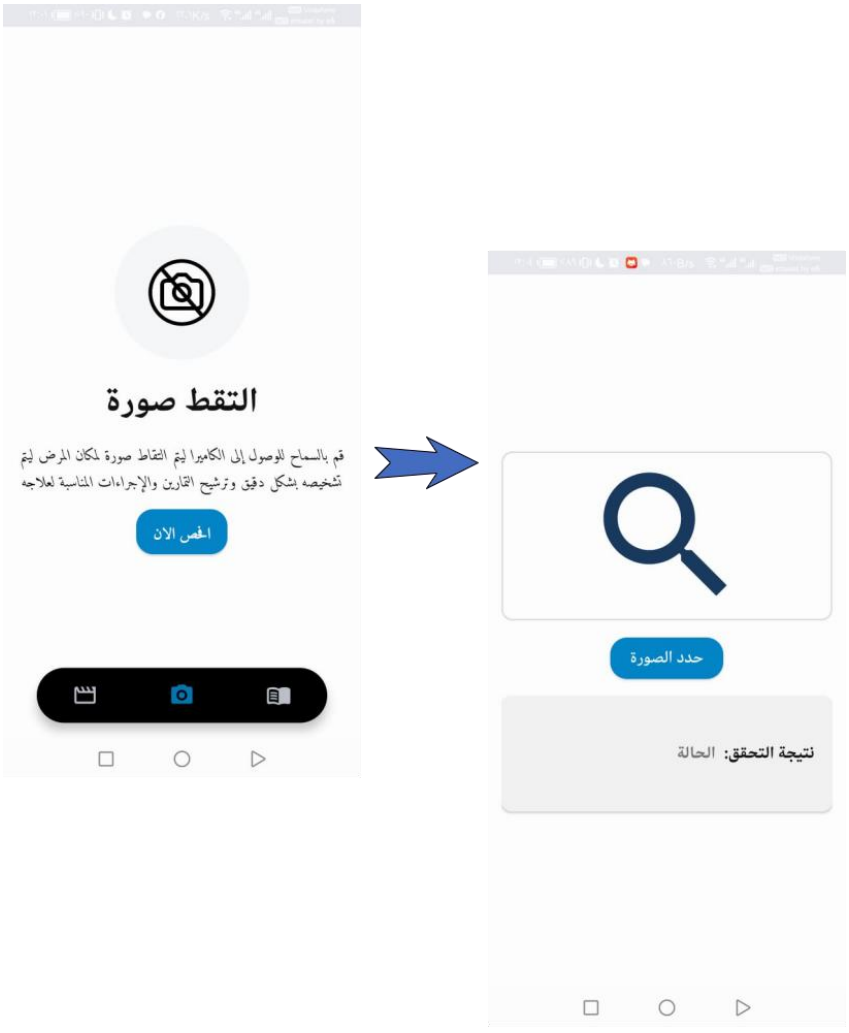


<< Fig 2.3 >>

Figure 2.4 choose Image for detect the case of patient and display the result

-Second screen of our app's home screen,which display through bottom navigation what's your coming navigation?

In the Home screen which display the button and text guide to pick the picture for person to check it and predict if it normal or not.



<< Fig 2.4 >>

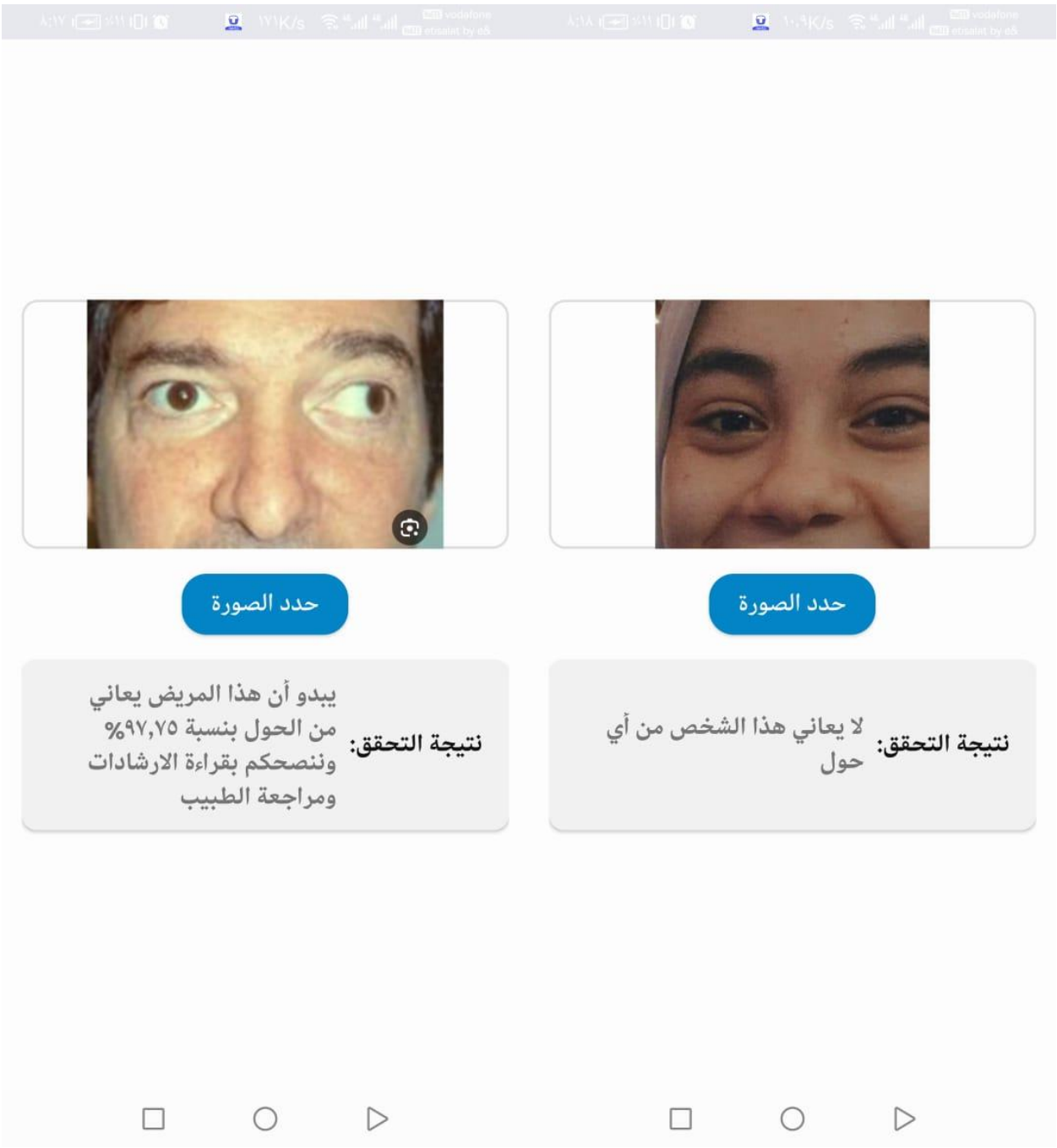
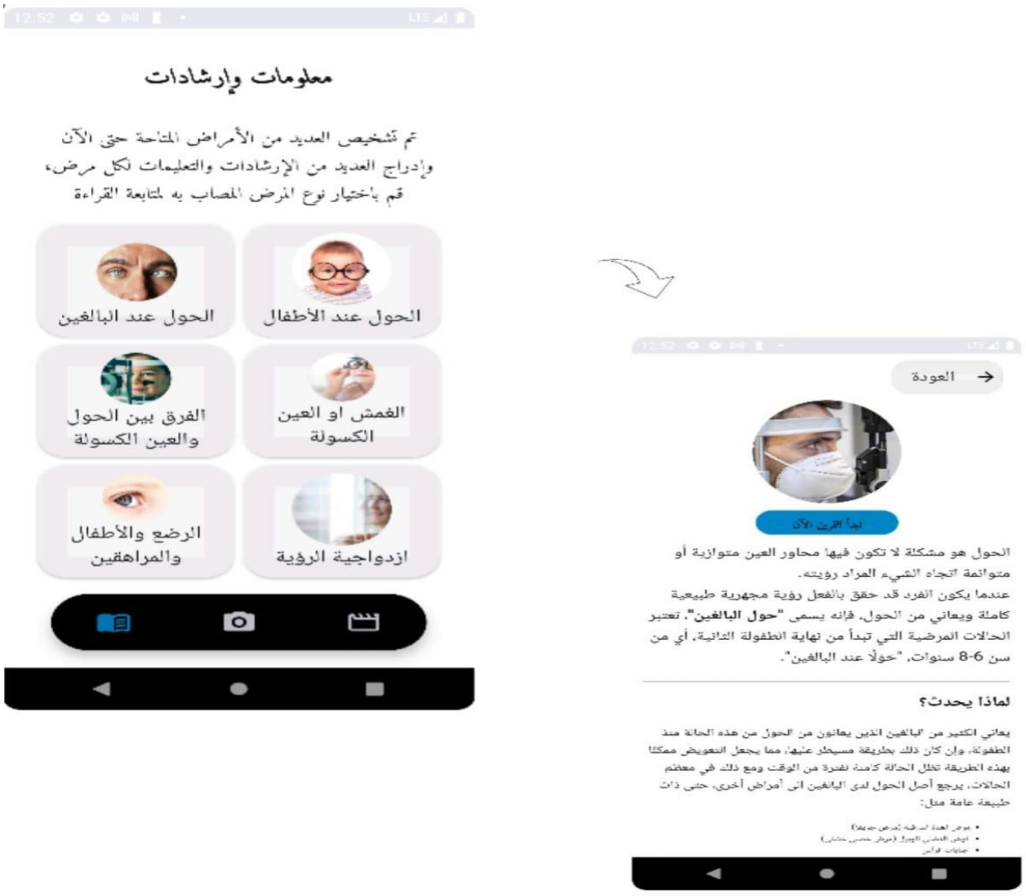

حدد الصورة
نتيجة التحقق:
لا يعاني هذا الشخص من أي حول
حدد الصورة
نتيجة التحقق:
يبدو أن هذا المريض يعاني من الحول بنسبة ٩٧,٧٥%
وننصحكم بقراءة الارشادات ومراجعة الطبيب

Figure 2.5 choose Articles and start to read it

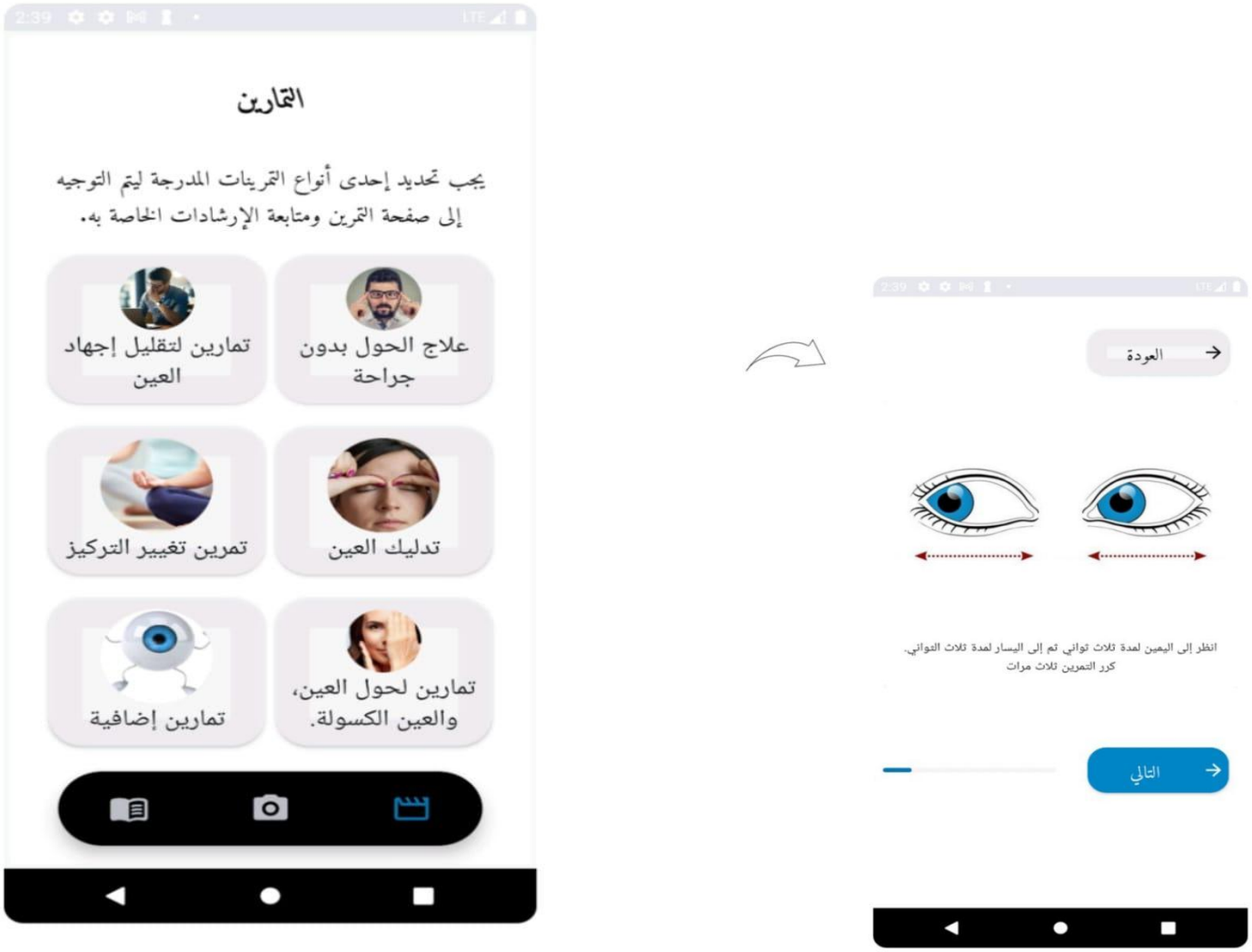
If user chooses it, it means that he want to enjoy one of features of this app , he can return to articles fragments or go to exercise

(figure 3.4).



<< Fig 2.5 >>

Figure 2.6 the 2nd feature of our app which guide user to instructions & exercises



<< Fig 2.6 >>

We can check the code for android implementation from this link http://github.com/asmaa2001-coder/GraduationProject

Chapter 4: System Implementation

Strabismus eyes detection model Implementation

4.1.1 Introduction

In this chapter we discuss the implementation and programming techniques that We use, we discuss how we convert the system from analysis state into real system That can be used by android users.

4.1.2 Strabismus Eye Detection implementation

SED system implementation divided into 2 parts, each part has its own

Steps:

4.1.2.1 Implementation of model:

1. Create dateset.
2. Choosing best algorithm to build the model.

2. Implementation of model.

a. Label images.
b. Prepossessing for input to model.
c. Testing the model.
d. Applying model.

4.1.2.2 Implementation the rest of the system:

1. Implement android.
2. offline integration

4.1.2.1.1 Create dateset:

We search for existing dateset but we not find so we create our own dateset.

Dateset contains 2 classes (strabismus or normal) each class has 504 images.

That's mean we have 1008 images in the file named Data

This images resizing to 224*224

Example image:

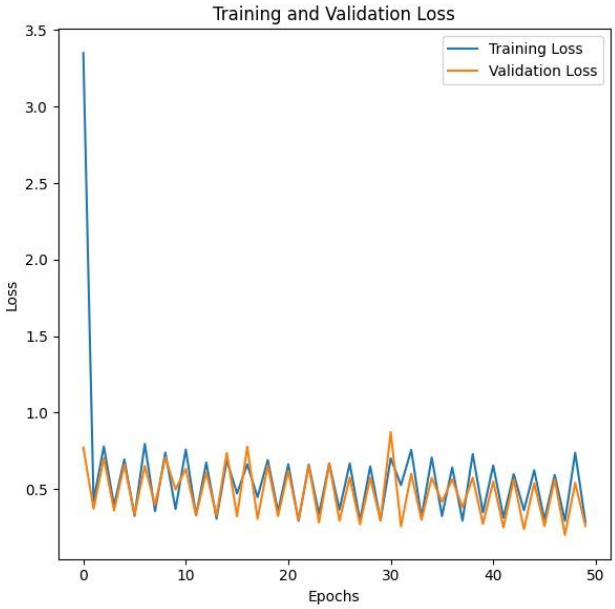
-strabismus -normal

4.1.2.1.2 Choosing best algorithm to build the Model:

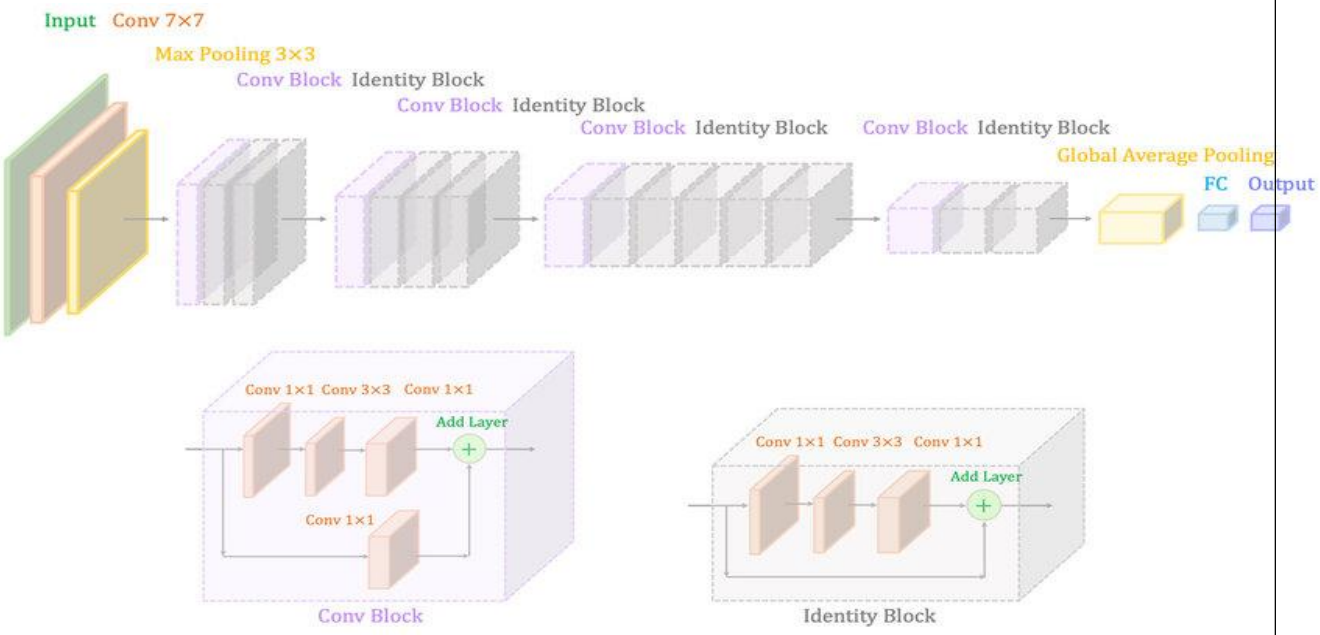
- **Proposed model**
  We try three models in our data
  - ✓ **First model is RESNET50 Model:**

  It gives 76% accuracy
  As shown in figure below

```
Epoch 36/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step - accuracy: 0.5938 - loss: 0.3232 - val_accuracy: 0.5238 - val_loss: 0.4164
Epoch 37/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 69s 3s/step - accuracy: 0.6019 - loss: 0.6417 - val_accuracy: 0.7437 - val_loss: 0.5644
Epoch 38/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step - accuracy: 0.7812 - loss: 0.2934 - val_accuracy: 0.5238 - val_loss: 0.3801
Epoch 39/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 69s 3s/step - accuracy: 0.5813 - loss: 0.7288 - val_accuracy: 0.6938 - val_loss: 0.5742
Epoch 40/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 53ms/step - accuracy: 0.5625 - loss: 0.3471 - val_accuracy: 0.7143 - val_loss: 0.2724
Epoch 41/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 68s 3s/step - accuracy: 0.6079 - loss: 0.6542 - val_accuracy: 0.7563 - val_loss: 0.5495
Epoch 42/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 50ms/step - accuracy: 0.6250 - loss: 0.3136 - val_accuracy: 0.8571 - val_loss: 0.2498
Epoch 43/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 69s 3s/step - accuracy: 0.6731 - loss: 0.5979 - val_accuracy: 0.7063 - val_loss: 0.5645
Epoch 44/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 58ms/step - accuracy: 0.5938 - loss: 0.3617 - val_accuracy: 0.9048 - val_loss: 0.2373
Epoch 45/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 115s 5s/step - accuracy: 0.6634 - loss: 0.6229 - val_accuracy: 0.7875 - val_loss: 0.5375
Epoch 46/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 47ms/step - accuracy: 0.7812 - loss: 0.3003 - val_accuracy: 0.6667 - val_loss: 0.2572
Epoch 47/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 62s 2s/step - accuracy: 0.6848 - loss: 0.5926 - val_accuracy: 0.6750 - val_loss: 0.5630
Epoch 48/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 3s 43ms/step - accuracy: 0.6562 - loss: 0.2915 - val_accuracy: 0.7619 - val_loss: 0.2002
Epoch 49/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 69s 3s/step - accuracy: 0.5553 - loss: 0.7378 - val_accuracy: 0.7750 - val_loss: 0.5405
Epoch 50/50
24/24 ━━━━━━━━━━━━━━━━━━━━ 4s 56ms/step - accuracy: 0.7188 - loss: 0.2842 - val_accuracy: 0.7619 - val_loss: 0.2579
```



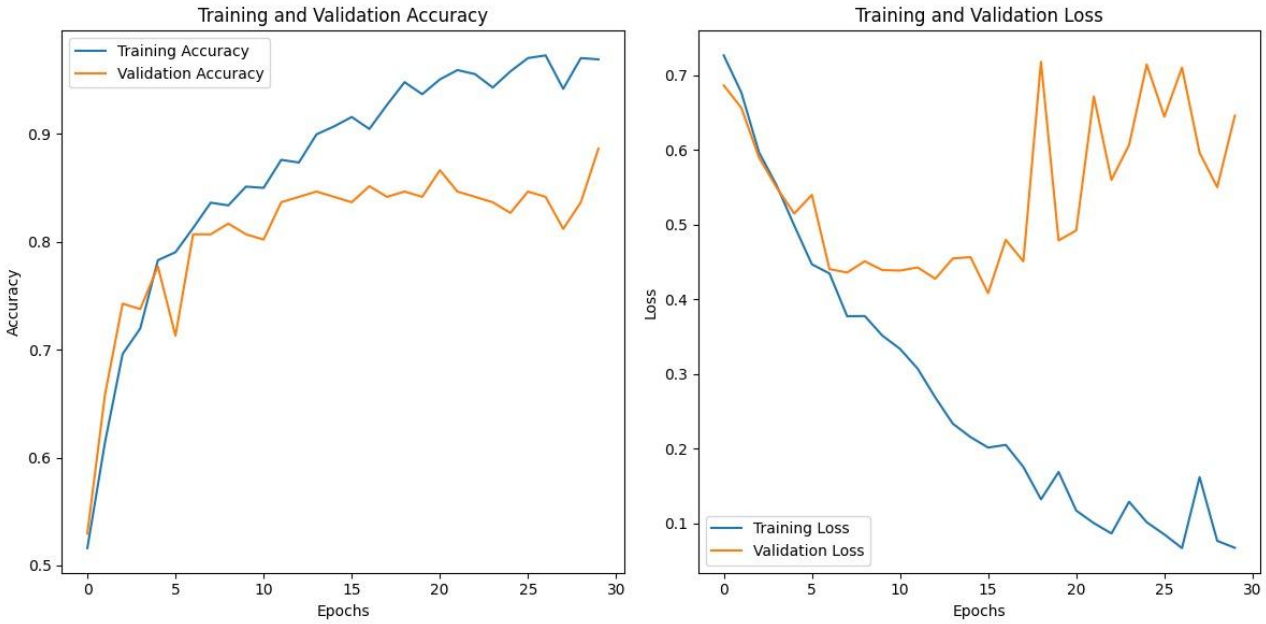
RESNET50 architecture :



✓ **Second model that we use is CNN:**

It give 88% accuracy

As shown in figure below



In this work, the proposed model uses the convolution neural network (CNN) to acquire deeply the features vector for automatic strabismus detection. The model is consisting of two stages: firstly, the eye region segmentation from the face is performed using eye detection model. Secondly, map the segmented eye regions into two output classes (strabismus: 1 or normal: 0) according to each eye iris position. The general flow diagram of the detection method illustrated in Figure 2

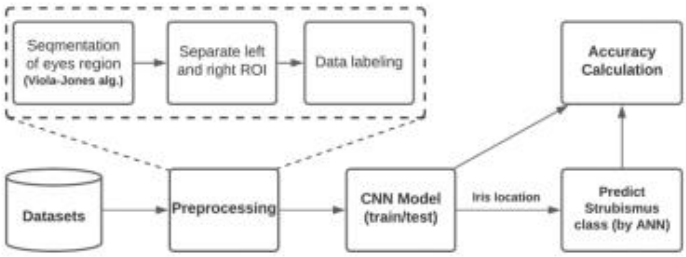
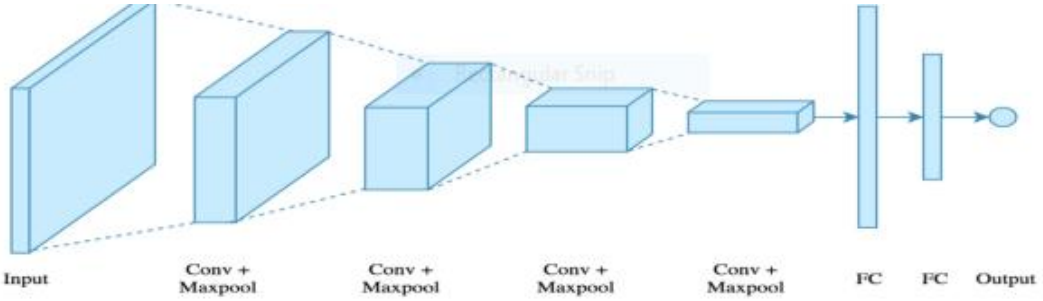


Figure 2. The general framework of the proposed algorithm

1.Convolution neural network:



Input is an images of pixel size—224×224.Let's understand the architecture of the model. The model contained 7 layers excluding the input layer, let's go layer by layer:

1.Layer 1: A convolution layer with kernel size of 5×5, stride of 1×1 and 6 kernels in total. So, the input image of size 32x32x1 gives an output of 28x28x6. Total params in layer = 5 * 5 * 6 + 6 (bias terms)

2. Layer 2: A pooling layer with 2×2 kernel size, stride of 2×2 and 6 kernels in total. This pooling layer acted a little differently than what we discussed in previous post. The input values in the receptive were summed up and then were multiplied to a trainable parameter (1 per filter), the result was finally added to a trainable bias (1 per filter). Finally, sigmoid activation was applied to the output. So, the input from previous layer of size 28x28x6 gets sub-sampled to 14x14x6. Total params in layer = [1 (trainable parameter) + 1 (trainable bias)] * 6 = 12

3.Layer 3: Similar to Layer 1, this layer is a convolutional layer with same configuration except it has 16 filters instead of 6. So, the input from previous layer of size 14x14x6 gives an output of 10x10x16. Total params in layer = 5 * 5 * 16 + 16 = 416.

4. Layer 4: Again, similar to Layer 2, this layer is a pooling layer with 16 filters this time around. Remember, the outputs are passed through sigmoid activation function. The input of size 10x10x16 from previous layer gets subsampled to 5x5x16. Total params in layer = (1 + 1) * 16 = 32

5. Layer 5: This time around we have a Convolution layer with 5×5 kernel size and 120 filters. There is no need to even consider strides as the input size is 5x5x16 so we will get an output of 1x1x120. Total params in layer = 5 * 5 * 120 = 3000

6. Layer 6: This is a dense layer with 84 parameters. So, the input of 120 units is converted to 84 units. Total params = 84 * 120 + 84 = 10164. The activation function used here was rather a unique one. I'll say you can just try out any of your choice here as the task is pretty simple one by today's standards.

7. Output Layer: Finally, a dense layer with 10 units is used. Total params = 84 * 10 + 10 = 924.

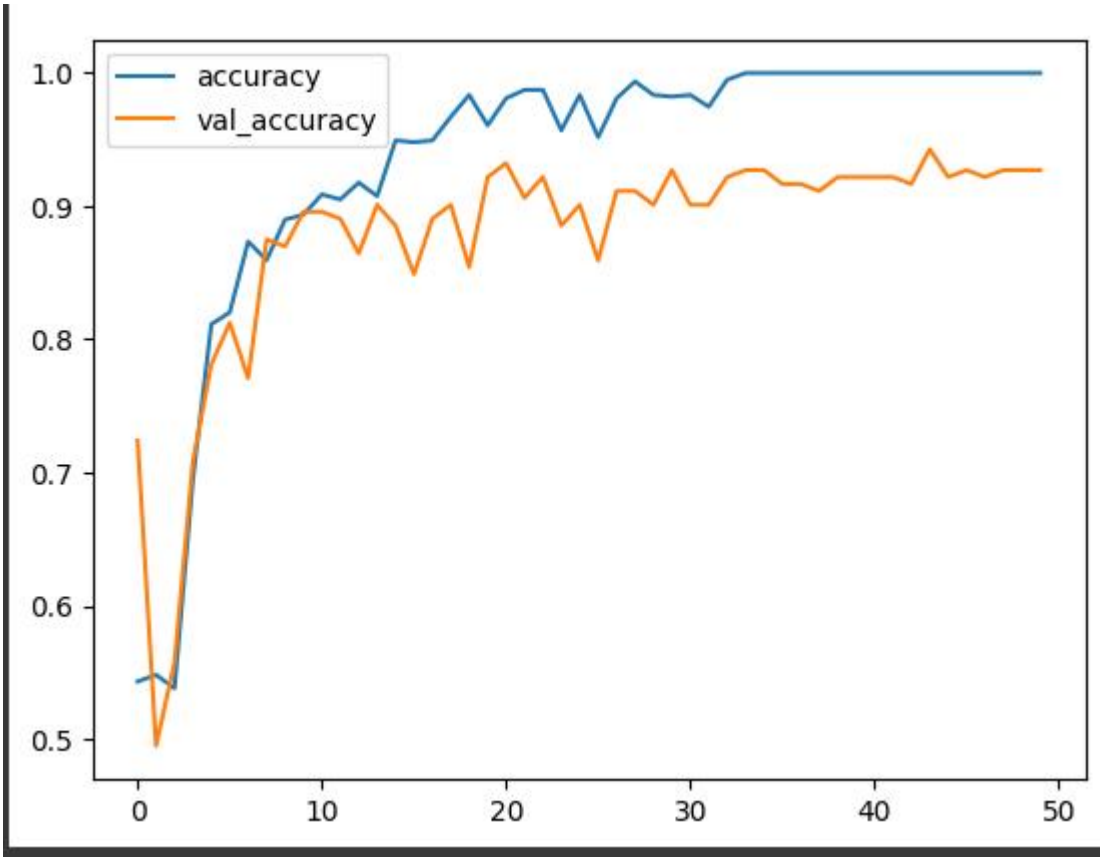
3-Thied model is vgg16:

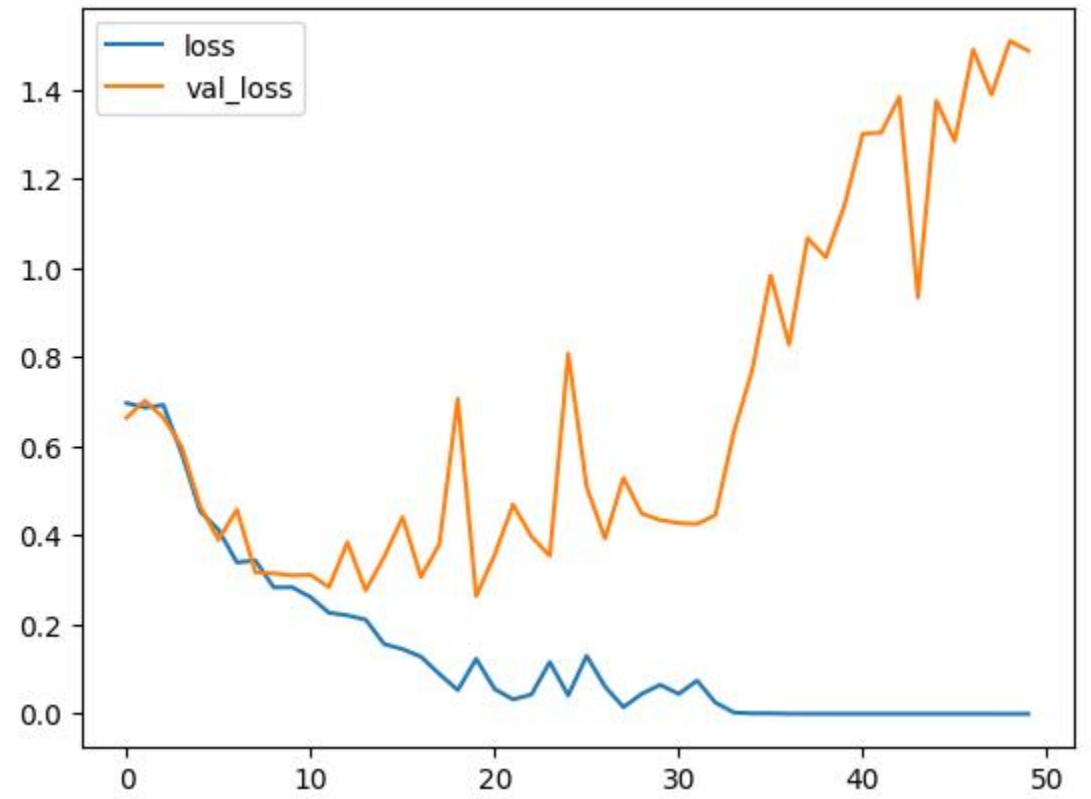
this is the best algorithm training on our data so we use it in our application

- It gives 92.57% accuracy as shown in figure below:

```
# Evaluate the model on the test data
loss, accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

```
13/13 [==============================] - 4s 301ms/step - loss: 1.5575 - accuracy: 0.9257
Test Accuracy: 92.57%
```





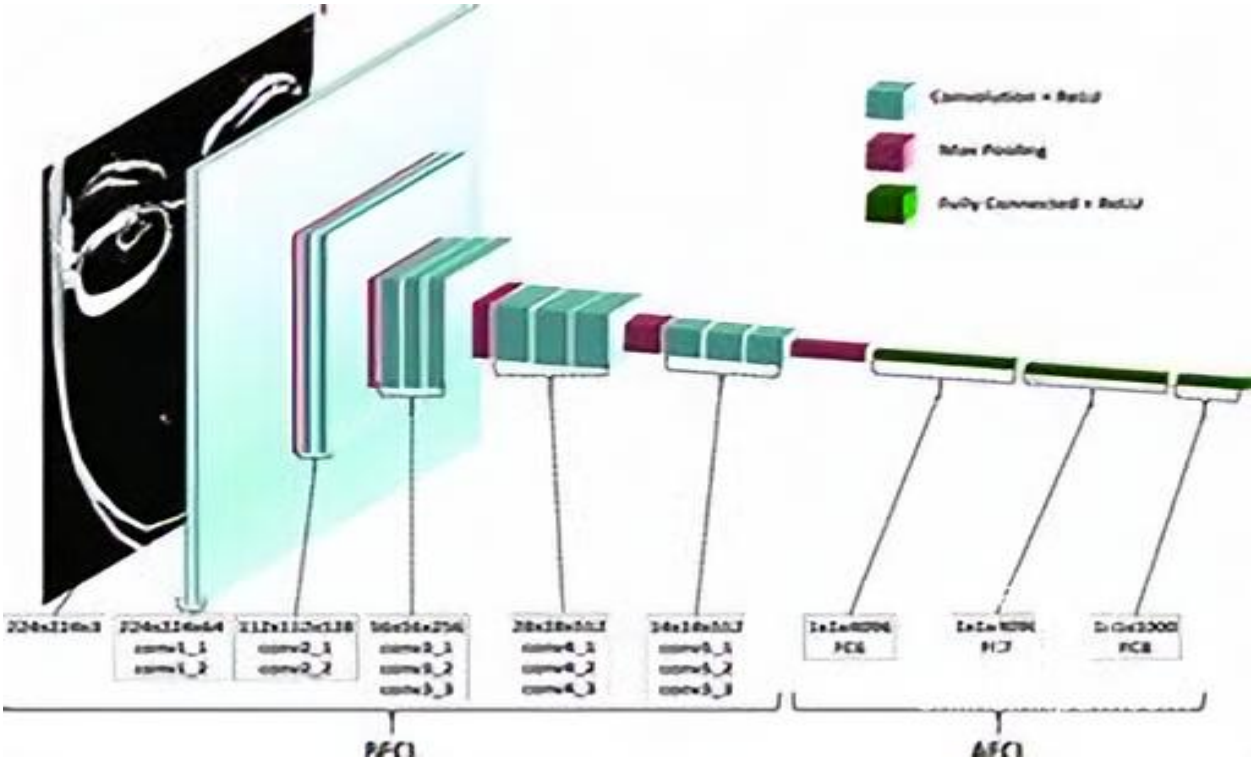
- Let us learn about VGG16 Model

VGG-16

The VGG-16 model is a convolutional neural network (CNN) architecture that was proposed by the Visual Geometry Group (VGG) at the University of Oxford. It is characterized by its depth, consisting of 16 layers, including 13 convolutional layers and 3 fully connected layers. VGG-16 is renowned for its simplicity and effectiveness, as well as its ability to achieve strong performance on various computer vision tasks, including image classification and object recognition. The model's architecture features a stack of convolutional layers followed by max-pooling layers, with progressively increasing depth. This design enables the model to learn intricate hierarchical representations of visual features, leading to robust and accurate predictions. Despite its simplicity compared to more recent architectures, VGG-16 remains a popular choice for many deep learning applications due to its versatility and excellent performance.

The ImageNet Large Scale Visual Recognition Challenge (ILSVRC) is an annual competition in computer vision where teams tackle tasks including object localization and image classification. VGG16, proposed by Karen Simonyan and Andrew Zisserman

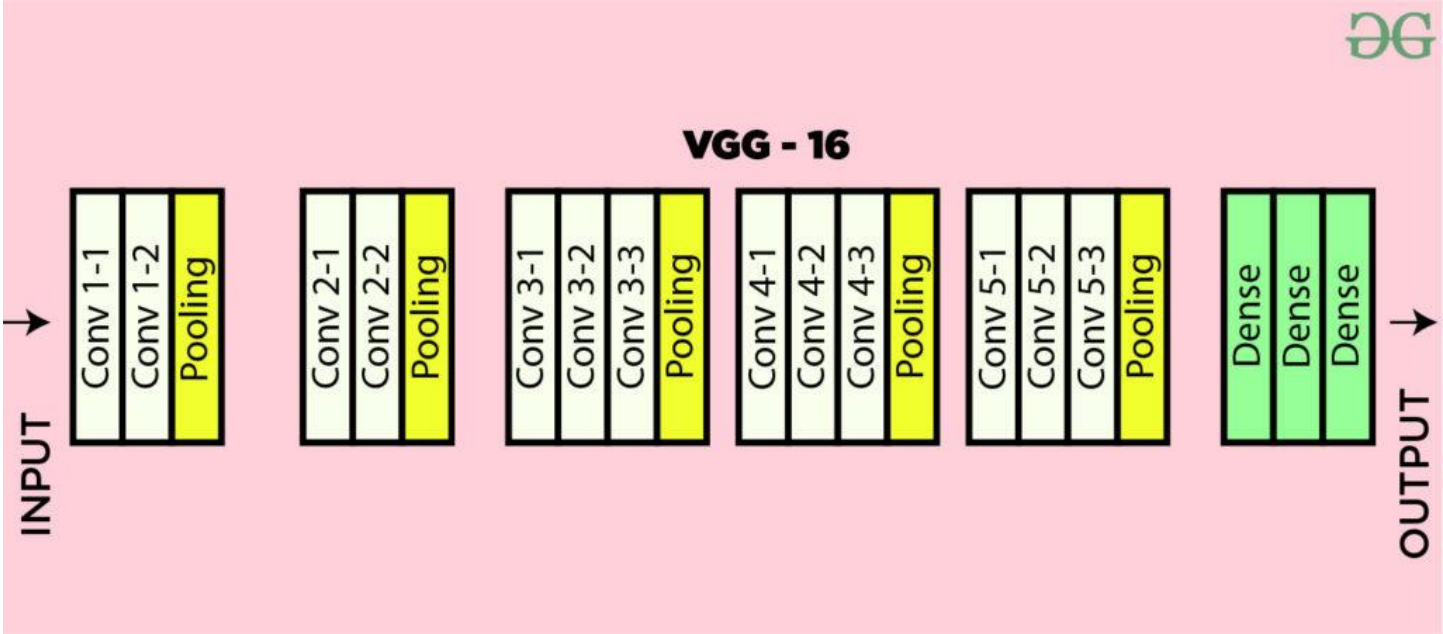
in 2014, achieved top ranks in both tasks, detecting objects from 200 classes and classifying images into 1000 categories



VGG Architecture:

The VGG-16 architecture is a deep convolutional neural network (CNN) designed for image classification tasks. It was introduced by the Visual Geometry Group at the University of Oxford. VGG-16 is characterized by its simplicity and uniform architecture, making it easy to understand and implement.

The VGG-16 configuration typically consists of 16 layers, including 13 convolutional layers and 3 fully connected layers. These layers are organized into blocks, with each block containing multiple convolutional layers followed by a max-pooling layer for downsampling.



Architecture Overview:

- VGG16 consists of a stack of convolutional layers followed by fully connected layers.
- It was developed by the Visual Geometry Group at the University of Oxford.
- The architecture is characterized by its simplicity and uniformity in design.
- It has a total of 16 weight layers, including 13 convolutional layers and 3 fully connected layers.

Convolutional Layers:

- The image is passed through a series of convolutional layers (conv. layers).
- Each conv. layer uses small receptive fields (3x3 filters) to capture local features.
- The filters learn to detect patterns such as edges, textures, and shapes.
- VGG16 stacks multiple conv. layers to learn increasingly complex features.

Pooling Layers:

- After each set of conv. layers, max-pooling layers downsample the feature maps.
- Max-pooling reduces spatial dimensions while retaining important features.

Fully Connected Layers:

- The final layers are fully connected (dense) layers.
- They combine high-level features to make predictions.
- In classification tasks, the output layer typically has as many neurons as the number of classes.

Application to Strabismus Detection:

- To apply VGG16 to strabismus detection, you'd need labeled eye images (including both normal and strabismic cases).
- Fine-tune the pre-trained VGG16 model on your dataset by adjusting the weights.
- Train the model to classify images as either normal or showing signs of strabismus.
- Evaluate the model's accuracy and performance.

This architecture follows the specifications provided, including the use of ReLU activation function and the final fully connected layer outputting probabilities for 2 classes using sigmoid activation.

Code Explaining:

1-Collecting Data:

```python
import os
import shutil
from sklearn.model_selection import train_test_split

# Define class names and the number of images per class
class_names = ["normal", "strabismus"]
class_image_counts = [504, 504]

# Base directory of the dataset
base_dir = '/content/dataset/Data'

# Create train and test directories
train_dir = os.path.join(base_dir, 'train')
test_dir = os.path.join(base_dir, 'test')

if not os.path.exists(train_dir):
    os.makedirs(train_dir)

if not os.path.exists(test_dir):
    os.makedirs(test_dir)
```

```python
# Move images into train and test directories
for class_name, image_count in zip(class_names, class_image_counts):
    class_dir = os.path.join(base_dir, class_name)
    images = os.listdir(class_dir)

    # Split the images into training and testing sets
    train_images, test_images = train_test_split(images, test_size=0.2, random_state=42)

    # Create class-specific directories in the train and test directories
    train_class_dir = os.path.join(train_dir, class_name)
    test_class_dir = os.path.join(test_dir, class_name)

    if not os.path.exists(train_class_dir):
        os.makedirs(train_class_dir)
    if not os.path.exists(test_class_dir):
        os.makedirs(test_class_dir)

    # Move images to their respective directories
    for image in train_images:
        src = os.path.join(class_dir, image)
        dst = os.path.join(train_class_dir, image)
        shutil.move(src, dst)

    for image in test_images:
        src = os.path.join(class_dir, image)
        dst = os.path.join(test_class_dir, image)
        shutil.move(src, dst)
```

It sets up the directory structure for your dataset, splits the images into training and testing sets, and organizes them by class (normal vs. strabismus). Let's break down what each part of our code does:

1-Class Names and Image Counts:

- we've defined two class names: "normal" and "strabismus."
- The corresponding image counts are 504 for each class.

2-Base Directory and Subdirectories:

- The base_dir variable points to the root directory where your dataset is stored.
- Inside this directory, you'll create subdirectories for training and testing data.

3-Train-Test Split:

- we use train_test_split from sklearn to split the images into training and testing sets.
- The test_size=0.2 argument means that 20% of the data will be used for testing.

4-Class-Specific Directories:

- For each class, we create directories within the train and test directories.
- These directories will hold the images for each class.

4-Moving Images:

- we move the images from their original class directories to the appropriate train or test directories.
- This ensures that the data is organized correctly for training and evaluation.

Import libraries

```
import tensorflow as tf
from tensorflow import keras
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications import VGG16
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Flatten, Dropout
from tensorflow.keras.optimizers import Adam
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
import os
from tensorflow.keras.applications.vgg16 import VGG16
from tensorflow.keras.layers import Dense, GlobalAveragePooling2D
from tensorflow.keras.models import Model
```

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

Data Generator:

```
# Define data generators with rescaling
train_datagen = ImageDataGenerator(rescale=1.0/255.0)
test_datagen = ImageDataGenerator(rescale=1.0/255.0)

# Define data directories
train_dir = '/content/dataset/Data/train'
test_dir = '/content/dataset/Data/test'

# Define batch size and image size
batch_size = 16
image_size = (224, 224)

# Load training data
train_generator = train_datagen.flow_from_directory(
    train_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary'
)

# Load testing data
test_generator = test_datagen.flow_from_directory(
    test_dir,
    target_size=image_size,
    batch_size=batch_size,
    class_mode='binary'
)
```

```
Found 806 images belonging to 2 classes.
Found 202 images belonging to 2 classes.
```

1-Data Generators:

- we've defined two data generators: train_datagen and test_datagen.
- These generators preprocess the images by rescaling pixel values to the range [0, 1].

2-Data Directories:

- we've specified the paths to the training and testing directories (train_dir and test_dir).
- These directories contain the organized images for each class.

3-Batch Size and Image Size:

- batch_size determines the number of images processed in each batch during training.
- image_size sets the target size for resizing the input images (224x224 pixels).

4-Loading Training and Testing Data:

- The flow_from_directory method loads data from the specified directories.
- For training, it creates batches of images from the training directory.
- For testing, it does the same from the testing directory.
- The class_mode='binary' indicates that you're working with binary classification (normal vs. strabismus).

❖Count and print the number of images in train and test directories:

```
import os
import glob as gb

# Count and print the number of images in train and test directories
for class_name in class_names:
    train_class_dir = os.path.join(train_dir, class_name)
    test_class_dir = os.path.join(test_dir, class_name)

    # Count images in the train directory
    train_files = gb.glob(pathname=os.path.join(train_class_dir, '*.jpg'))
    print(f'For training data, found {len(train_files)} images in class {class_name}')

    # Count images in the test directory
    test_files = gb.glob(pathname=os.path.join(test_class_dir, '*.jpg'))
    print(f'For testing data, found {len(test_files)} images in class {class_name}')
```

```
For training data, found 403 images in class normal
For testing data, found 101 images in class normal
For training data, found 403 images in class strabismus
For testing data, found 101 images in class strabismus
```

Image sizes in the training datatset:

```
import pandas as pd

train_sizes = []  # List to store image sizes in the training dataset

# Iterate through each class in the training directory
for class_name in os.listdir(train_dir):
    train_class_dir = os.path.join(train_dir, class_name)

    # Iterate through each image in the class
    for file in gb.glob(pathname=os.path.join(train_class_dir, '*.jpg')):
        image = plt.imread(file)
        train_sizes.append(image.shape)

# Create a Pandas Series to count and display the unique image sizes
train_size_counts = pd.Series(train_sizes).value_counts()

# Print the counts of unique image sizes in the training dataset
print("Image sizes in the training dataset:")
print(train_size_counts)
```

```
Image sizes in the training dataset:
(224, 224, 3)    806
Name: count, dtype: int64
```

Image sizes in the testing datatset:

```
test_sizes = []  # List to store image sizes in the testing dataset

# Iterate through each class in the training directory
for class_name in os.listdir(test_dir):
    test_class_dir = os.path.join(test_dir, class_name)

    # Iterate through each image in the class
    for file in gb.glob(pathname=os.path.join(test_class_dir, '*.jpg')):
        image = plt.imread(file)
        test_sizes.append(image.shape)

# Create a Pandas Series to count and display the unique image sizes
test_size_counts = pd.Series(test_sizes).value_counts()

# Print the counts of unique image sizes in the training dataset
print("Image sizes in the testing dataset:")
print(test_size_counts)
```

```
Image sizes in the testing dataset:
(224, 224, 3)    202
Name: count, dtype: int64
```

Load the Model:

```
# Load pre-trained VGG16 model (excluding top layers)
base_model = VGG16(weights='imagenet', include_top=False, input_shape=(224, 224, 3))

# Add custom classification head
x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(512, activation='relu')(x)
predictions = Dense(1, activation='sigmoid')(x)  # Binary classification (strabismus or not)

model = Model(inputs=base_model.input, outputs=predictions)
```

Compile the Model:

```
model.compile(optimizer=Adam(learning_rate=0.0001), loss='binary_crossentropy', metrics=['accuracy'])
```

The model.compile step is essential for configuring our model before training. Let's break down what this line does:

1-Optimizer:

- we've chosen the Adam optimizer with a learning rate of 0.0001.
- The optimizer adjusts the model's weights during training to minimize the loss.

2-Loss Function:

- The loss function we've specified is 'binary_crossentropy'.
- This loss measures the difference between predicted and actual values for binary classification tasks.

3-Metrics:

- we're tracking the accuracy metric during training.
- Accuracy indicates how well the model predicts the correct class labels.

Applying Model on the data:

```
history = model.fit(
    train_generator,
    steps_per_epoch=train_generator.samples // train_generator.batch_size,
    validation_data=test_generator,
    validation_steps=test_generator.samples // test_generator.batch_size,
    epochs=50
)
```

```
Epoch 22/50
50/50 [==============================] - 12s 238ms/step - loss: 0.0324 - accuracy: 0.9873 - val_loss: 0.4694 - val_accuracy: 0.9062
Epoch 23/50
50/50 [==============================] - 12s 229ms/step - loss: 0.0432 - accuracy: 0.9873 - val_loss: 0.3975 - val_accuracy: 0.9219
Epoch 24/50
50/50 [==============================] - 12s 242ms/step - loss: 0.1161 - accuracy: 0.9570 - val_loss: 0.3545 - val_accuracy: 0.8854
Epoch 25/50
50/50 [==============================] - 12s 239ms/step - loss: 0.0412 - accuracy: 0.9835 - val_loss: 0.8079 - val_accuracy: 0.9010
Epoch 26/50
50/50 [==============================] - 12s 243ms/step - loss: 0.1301 - accuracy: 0.9519 - val_loss: 0.5097 - val_accuracy: 0.8594
Epoch 27/50
50/50 [==============================] - 12s 235ms/step - loss: 0.0610 - accuracy: 0.9810 - val_loss: 0.3930 - val_accuracy: 0.9115
Epoch 28/50
50/50 [==============================] - 12s 240ms/step - loss: 0.0150 - accuracy: 0.9937 - val_loss: 0.5293 - val_accuracy: 0.9115
Epoch 29/50
50/50 [==============================] - 12s 228ms/step - loss: 0.0449 - accuracy: 0.9835 - val_loss: 0.4493 - val_accuracy: 0.9010
Epoch 30/50
50/50 [==============================] - 12s 232ms/step - loss: 0.0652 - accuracy: 0.9823 - val_loss: 0.4344 - val_accuracy: 0.9271
Epoch 31/50
50/50 [==============================] - 12s 240ms/step - loss: 0.0446 - accuracy: 0.9835 - val_loss: 0.4278 - val_accuracy: 0.9010
Epoch 32/50
50/50 [==============================] - 12s 243ms/step - loss: 0.0745 - accuracy: 0.9747 - val_loss: 0.4259 - val_accuracy: 0.9010
Epoch 33/50
50/50 [==============================] - 12s 229ms/step - loss: 0.0252 - accuracy: 0.9949 - val_loss: 0.4459 - val_accuracy: 0.9219
Epoch 34/50
50/50 [==============================] - 12s 234ms/step - loss: 0.0030 - accuracy: 1.0000 - val_loss: 0.6319 - val_accuracy: 0.9271
Epoch 35/50
50/50 [==============================] - 12s 243ms/step - loss: 9.1188e-04 - accuracy: 1.0000 - val_loss: 0.7711 - val_accuracy: 0.9271
Epoch 36/50
```

we've initiated the training process for our strabismus detection model using the VGG16 architecture. Let's break down what our code does:

1-Training Process:

- The model.fit function trains our model using the training data (train_generator).
- It specifies the number of steps per epoch (based on the batch size) and the validation data (test_generator).
- The epochs parameter determines how many times the model will iterate over the entire dataset during training.

2-Monitoring Progress:

- As the model trains, it computes the loss and accuracy on both the training and validation sets.

- we’ll see updates for each epoch, including training loss, training accuracy, validation loss, and validation accuracy.

Model Evaluation:

```
# Evaluate the model on the test data
loss, accuracy = model.evaluate(test_generator)
print(f"Test Accuracy: {accuracy * 100:.2f}%")
```

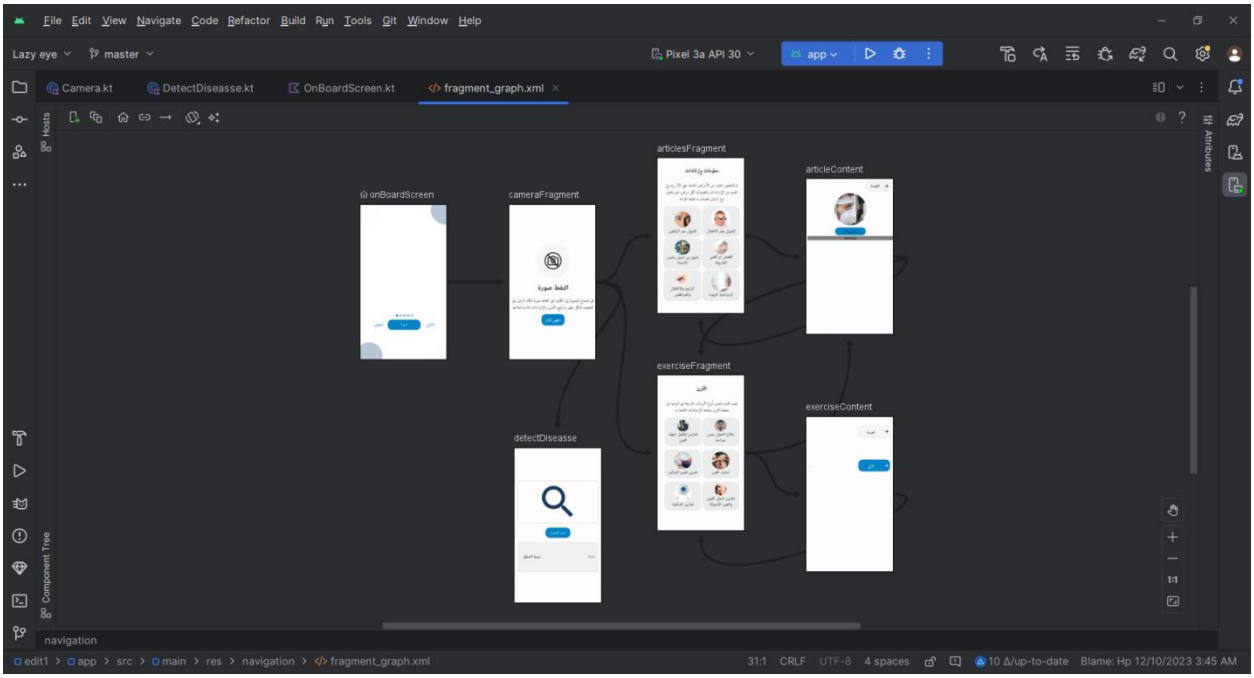
```
13/13 [==============================] - 4s 301ms/step - loss: 1.5575 - accuracy: 0.9257
Test Accuracy: 92.57%
```

Save the Model:

```
model.save('eye_disease_model.h5')
```

4.1.2.2 Implementation the rest of the system:

In the same time when the AI engineers creates the model , Android Engineer start to create the Front-End through using the concept of Fragments to make it more light and easier to navigate from view to another



4.1.2.2.I Implementation the on Board Screen:

We will use the View pager to implement the guide to use the application after preparing the XML file for it and using "**shared preference**" to check if user use the app for first time of not

4.1.2.2.II Implementation the on Article Feature:

In this feature we will divide it into 2 Tasks :

- Prepare Articles Screen with using **"card view"** to show the list of articles.
- Prepare the content of the article ,we use for the article "**WEB-VIEW**" to make article in the format which make it suitable and desirable for reading by user so we will use **HTML** to write article.
- We use the "**RoomDB"** to save the content of the article ,this technique is easier and less complex from SQLite.

```
@Dao
interface DataDao {
    hp
    @Insert (onConflict = OnConflictStrategy.REPLACE)
    fun addAllExercise(exercise: List<Data>)

    hp
    @Delete
    fun delete(exercise: List<Data>)

    hp
    @Query("SELECT * FROM Data")
    fun getAllExercise(): List<Data>
}
```

- Page Adapter to connect the content of DB with view:

```
class PageAdapter(private val listOfData: List<Data> , val context: Context , private val resource:Int) :
    RecyclerView.Adapter<PageAdapter.PageViewHolderAdapter>() {

    hp
    inner class PageViewHolderAdapter(itemView: View) : RecyclerView.ViewHolder(
        itemView
    ) {
        val exDetails: TextView = itemView.findViewById(R.id.exer_guide)
        val imageDetail: ImageView = itemView.findViewById(R.id.exer_image)

    }

    hp
    override fun onCreateViewHolder(parent: ViewGroup , viewType: Int): PageViewHolderAdapter {
        return PageViewHolderAdapter(
            LayoutInflater.from(parent.context).inflate(resource , parent , attachToRoot: false)
        )
    }

    hp
    override fun getItemCount()=listOfData.size

    hp
    override fun onBindViewHolder(holder: PageViewHolderAdapter , position: Int) {
        val item = listOfData[position]
        holder.exDetails.text=item.des
        Glide.with(context) RequestManager
                .load(Uri.parse( uriString: "file:///android_asset/${item.image}")) RequestBuilder<Drawable!>
                .into(holder.imageDetail)       // holder.image_detail.setImageResource(item.image)

    }
}
```

4.1.2.2.III Implementation the on Exercise Feature:

- Prepare Articles Screen with using **“card view”** to show the list of Exercises.
- Prepare the content of the Exercise,we use for the Exercise“**Pager-View**” to make exercise in the format which make it suitable and desirable for user.

4.1.2.2.V Implementation the on Camera Feature:

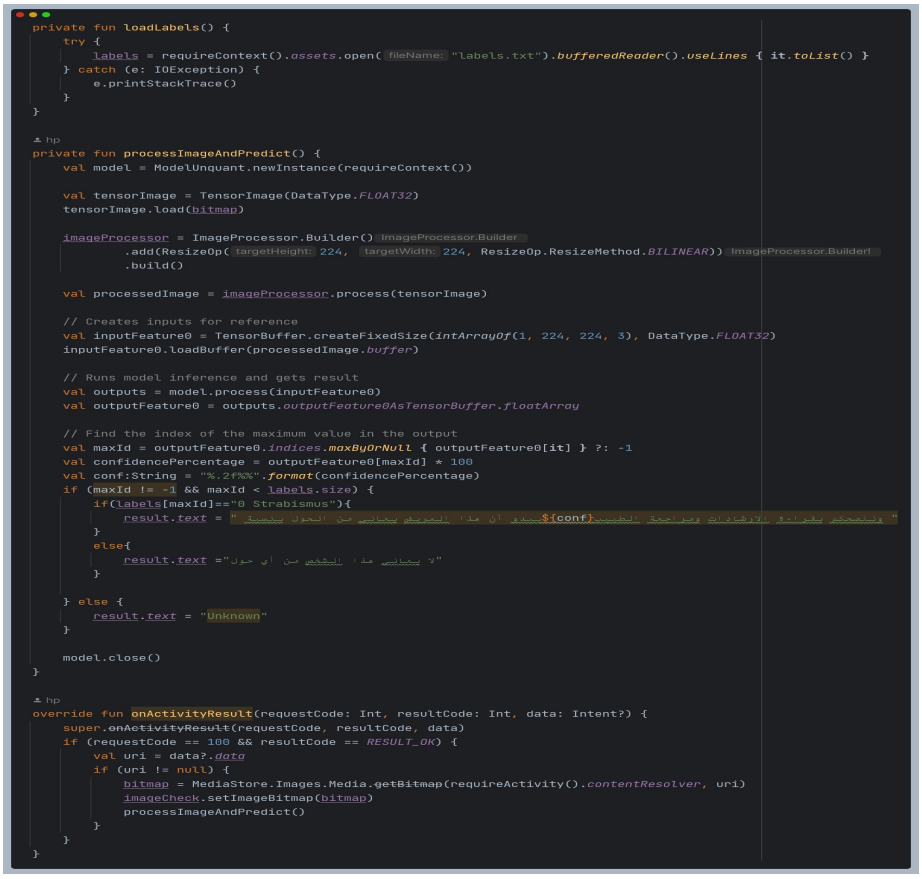
- Prepare Camera Screen .
- Prepare the page of the checking ,user will select image from Gallery or Drive to check it and display the result in the Text-View with the confidence
-

4.1.2.2.IV Implementation the Integration between AI model and Android:

Preferring to use the offline integration to make it suitable for the user’s infrastructure and can use it in anytime without connecting to Network.

We should prepare:

- Labels
- Processing on image and feeding it to the model



```
private fun loadLabels() {
    try {
        labels = requireContext().assets.open( fileName: "labels.txt").bufferedReader().useLines { it.toList() }
    } catch (e: IOException) {
        e.printStackTrace()
    }
}

hp
private fun processImageAndPredict() {
    val model = ModelUnquant.newInstance(requireContext())

    val tensorImage = TensorImage(DataType.FLOAT32)
    tensorImage.load(bitmap)

    imageProcessor = ImageProcessor.Builder() ImageProcessor.Builder
            .add(ResizeOp( targetHeight: 224,  targetWidth: 224, ResizeOp.ResizeMethod.BILINEAR)) ImageProcessor.Builder!
            .build()

    val processedImage = imageProcessor.process(tensorImage)

    // Creates inputs for reference
    val inputFeature0 = TensorBuffer.createFixedSize(intArrayOf(1, 224, 224, 3), DataType.FLOAT32)
    inputFeature0.loadBuffer(processedImage.buffer)

    // Runs model inference and gets result
    val outputs = model.process(inputFeature0)
    val outputFeature0 = outputs.outputFeature0AsTensorBuffer.floatArray

    // Find the index of the maximum value in the output
    val maxId = outputFeature0.indices.maxByOrNull { outputFeature0[it] } ?: -1
    val confidencePercentage = outputFeature0[maxId] * 100
    val conf:String = "%.2f%%".format(confidencePercentage)
    if (maxId != -1 && maxId < labels.size) {
        if(labels[maxId]=="0 Strabismus"){
            result.text = " ${conf}"
        }
        else{
            result.text ="لا يمكنني التنبؤ من اي حول"
        }

    } else {
        result.text = "Unknown"
    }

    model.close()
}

hp
override fun onActivityResult(requestCode: Int, resultCode: Int, data: Intent?) {
    super.onActivityResult(requestCode, resultCode, data)
    if (requestCode == 100 && resultCode == RESULT_OK) {
        val uri = data?.data
        if (uri != null) {
            bitmap = MediaStore.Images.Media.getBitmap(requireActivity().contentResolver, uri)
            imageCheck.setImageBitmap(bitmap)
            processImageAndPredict()
        }
    }
}
```

Chapter 5: Future Work

5.1 Expansion of Informational Resources

In future iterations of the Strabismus Eye Detection System, we plan to significantly expand the database of informational resources. This will include:

- **Additional Articles**: Increasing the quantity and depth of articles on strabismus, various eye diseases, and general eye care practices. The goal is to provide a comprehensive resource for users seeking to learn more about eye health.
- **Multimedia Content**: Integrating videos, interactive infographics, and other multimedia content to enhance user engagement and understanding.

5.2 Enhanced Connectivity with Healthcare Professionals

Future versions of the application will focus on establishing connections between users and healthcare professionals:

- **Doctor Consultation Feature**: Introducing a feature that allows users to connect with ophthalmologists and other eye care specialists for professional consultations. This could include telemedicine capabilities, appointment scheduling, and direct messaging.
- **Referral System**: Implementing a referral system to guide users towards trusted eye care providers based on their location and specific needs.

5.3 Development of Additional Detection Models

Expanding the machine learning capabilities of the application to detect a wider range of eye diseases:

- **New Models for Different Eye Diseases**: Developing and integrating models to detect various eye conditions such as glaucoma, cataracts, and macular degeneration. This will involve curating datasets for each condition, training models, and validating their accuracy.
- **Improved Accuracy and Precision**: Continuously improving the existing strabismus detection model to enhance its accuracy and reliability. This will include refining algorithms, increasing the dataset size, and leveraging more advanced machine learning techniques.

5.4 User Experience and Interface Enhancements

Ongoing improvements to the user interface and overall user experience:

- **UI/UX Refinements**: Regular updates to the app's design based on user feedback, with a focus on making the interface more intuitive and accessible.
- **Accessibility Features**: Ensuring the app is accessible to users with disabilities, including those with visual impairments.

5.5 Integration with Wearable Technology

Exploring the integration of the application with wearable devices:

- **Smart Glasses and Eye Trackers**: Investigating the potential to connect with smart glasses and eye-tracking devices to provide real-time monitoring and detection of eye conditions.
- **Health Data Synchronization**: Allowing synchronization with other health apps and wearables to provide a more holistic view of the user's health and eye care needs.

5.6 Internationalization and Localization

Making the application accessible to a global audience:

- **Language Support**: Adding support for multiple languages to cater to non-English speaking users.
- **Cultural Adaptation**: Customizing content to reflect cultural differences in eye care practices and healthcare systems.

5.7 Ethical and Regulatory Compliance

Ensuring the application adheres to ethical standards and regulatory requirements:

- **Data Privacy and Security**: Continuously enhancing data protection measures to safeguard user information.
- **Compliance with Healthcare Regulations**: Ensuring the application complies with relevant healthcare regulations and standards, such as HIPAA in the United States and GDPR in Europe

Conclusion

This Android app harnesses the power of computer vision to make strabismus detection more accessible. By providing an easy-to-use tool for analyzing eye images, the app can help identify strabismus early and connect users to the care they need. We believe this app has the potential to make a real difference in managing this common eye condition.