TEAM_04

API documentation:

In this setup, we've built the app's front and back ends in Android Studio and integrated the app's model into the code using the model. In order to communicate with the chatbot, we've implemented a Tflite and chat gpt/postman to interface API.

Introduction:-

This application programming interface (API) provides a means through which COVID-19 may be detected in a picture by employing AI methods. The artificial intelligence algorithm has been trained on a huge dataset of chest X-ray and CT scan pictures to identify the telltale signs of COVID-19 infection.

Kaggle dataset:-

Kaggle is widely used by data scientists and machine learning engineers as a repository for datasets of all kinds. Kaggle makes its datasets available through a public API. Developers may use this API to look for, get, and provide other people access to datasets.

Authentication:-

Kaggle account authentication is required to utilize the Datasets API. Kaggle's account settings allow you to produce an API token with an API key and secret. Your API queries need these credentials.

Endpoint The endpoint for the Kaggle Datasets API is

https://www.kaggle.com/api/v1.

Search for Datasets To search for datasets, you can use the following endpoint:

GET /datasets/list?sort=hotness&size=20&filetype=csv&q=covid

Example Request:

{
"total": 10,
"datasets": [

{ "ref": "imdevskp/corona-virus-report", "title": "Coronavirus (COVID-19) Report - Latest Data", "subtitle": "Daily updates on confirmed cases, deaths and recoveries.", "is_public": true,
"last_updated": "2022-04-28T12:00:00Z",
"download_count": 100000,

"size": 1024,
"owner": { "username": "hanith9695", "display_name": "Hanith", "is_organization": false,
"avatar_url": "https://www.kaggle.com/avatars/imdevskp.jpg"
},
"file_type": "csv"
}
]
}

CameraPredictionActivity.java:-

1. **Class CameraPredictionActivity**

This method takes a picture from the device's camera and analyzes it with a machine learnin g model that has been trained to determine whether or not the image includes COVID-19.

Public Methods:

2. **onCreate(Bundle savedInstanceState): void**

This function is invoked just after the activity has been created. The UI components are initia lized, the machine learning model is loaded, and event listeners are set up.

3. **openCamera(): void**

When a user presses the camera button, this function is invoked.
It verifies the user has provided the necessary rights before launching the camera.

4. **onRequestPermissionsResult(int requestCode, @NonNull String[] permissions, @NonNull int[] grantResults): void**

When the user grants permission, this function is invoked. If the necessary permissions are a vailable, it will open the device's camera to take a picture.

5. **onActivityResult(int requestCode, int resultCode, Intent data): void**

Once the image has been captured, the camera activity will call this method.
It stores the picture's URL in a member variable and shows the image on the user interface.

6. **loadModelFile(Activity activity): MappedByteBuffer**

The predictive model for machine learning is read in from the model file located in the asset s folder.

7. **loadImage(Bitmap bitmap): TensorImage**

The taken picture is stored in a TensorImage object after being loaded using this method.

8. **showResult(): void**

This technique takes a picture, runs it through a machine learning model that has been load ed, and then shows the result on the user interface.

Private Members:

1. **Interpreter mtfLite**

To execute the downloaded machine learning model, a TensorFlow Lite interpreter is used.

2. **int mImageSizeX**

The minimum width for a machine learning model to properly process an input picture.

3. **int mImageSizeY**

The minimum width for a valid input picture when using an ML algorithm.

4. **int mImageSizeY**

The needed picture height for the machine learning model's input.

5. **TensorImage mInputImageBuffer**

A TensorImage object used for holding the input image to be processed by the machine learning model.

6. **TensorBuffer mOutputProbabilityBuffer**

A TensorBuffer object used for holding the output probability values generated by the machine learning model.

7. **TensorProcessor mProbabilityProcessor**

A TensorProcessor object used for processing the output probability values generated by the machine learning model.

8. **public static final float IMAGE_MEAN**

The mean value to be subtracted from the input image pixels for normalization.

9. **public static final float IMAGE_STD**

The standard deviation value to be divided by the input image pixels for normalization.

10. **public static final float PROBABILITY_MEAN**

The mean value to be subtracted from the output probability values for post-processing.

11. **public static final float PROBABILITY_STD**

The standard deviation value to be divided by the output probability values for post-processing.

12. **static final int PERMISSION_CODE**

The request code used for requesting camera and storage permissions.

13. **static final int IMAGE_CAPTURE_CODE**

The request code used for capturing an image using the device's camera.

14. **Bitmap mBitmap**

The captured image in Bitmap format.

15. **Uri mImageUri**

The URI of the captured image.

16. **List<String> mLabels**

A list of labels used for classifying the input image into COVID-19 and non-COVID-19 classes.

17. **ImageView mImageView**

The ImageView object used for displaying the captured image on the UI.

18. **Button mButtonPredict**

The Button object used for initiating the image classification process.

19. **TextView mPredictText**

The TextView object used for displaying the classification result on the UI.

Class: BotActivity:-

Description:

This class represents the main activity of the chatbot application. It handles user input, sends it to the Dialogflow API for processing, and displays the response to the user.

Methods:

1**. onCreate(Bundle savedInstanceState)**

This method is called when the activity is first created. It initializes the chat view, the edit message field, and the send button. It also sets up the Dialogflow API by calling `setUpBot()` method.

2. **setUpBot()**

This method sets up the Dialogflow API by creating a session with a unique ID, retrieving the credentials from the credential file, and creating a SessionsClient object to send messages to the API.

3. **sendMessageToBot(String message)**

This method sends a message to the Dialogflow API for processing. It creates a QueryInput object with the user's message and sends it to the API using the SessionsClient object.

4. **callback(DetectIntentResponse returnResponse)**

This method is called when the Dialogflow API responds with a message. It retrieves the response message from the DetectIntentResponse object and displays it to the user in the chat view.

Variables:

1. RecyclerView chatView

This variable holds the reference to the RecyclerView object that displays the chat messages.

2. ChatAdapter chatAdapter

This variable holds the reference to the ChatAdapter object that manages the chat messages.

3. List<Message> messageList

This variable holds the list of chat messages.

4. EditText editMessage

This variable holds the reference to the EditText object where the user enters their messages.

5. ImageButton btnSend

This variable holds the reference to the ImageButton object that the user clicks to send their message.

6. SessionsClient sessionsClient

This variable holds the reference to the SessionsClient object that sends messages to the Dialogflow API.

7. SessionName sessionName

This variable holds the reference to the SessionName object that represents the session ID used to communicate with the Dialogflow API.

8. String uuid

This variable holds the unique ID of the session used to communicate with the Dialogflow API.

9. String TAG

This variable holds the tag used for logging messages in the application.
This is a simple activity class that displays the layout of the contact page in the LET it free application.

Methods:

1. **onCreate(Bundle savedInstanceState):**

Called when the activity is starting. This method initializes the activity by inflating the layout of the contact page.

Parameters:

1. **savedInstanceState:**

a Bundle object containing the activity's previously saved state.

Returns: void

Example:

```
public class ContactActivity extends AppCompatActivity {

   @Override
   protected void onCreate(Bundle savedInstanceState) {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.activity_contact);
   }
}
```

GalleryPredictionActivity.java:

Description:-

This class allows users to select an X-RAY image from their phone gallery and make a prediction using a pre-trained TensorFlow Lite model.

Class Hierarchy

```
java.lang.Object
   android.content.Context
      android.content.ContextWrapper
         android.view.ContextThemeWrapper
            androidx.appcompat.app.AppCompatActivity
```

Class Declaration

java
public class GalleryPredictionActivity extends AppCompatActivity

Constants:-

java
public static final float IMAGE_MEAN = 0.0f;
public static final float IMAGE_STD = 1.0f;
public static final float PROBABILITY_MEAN = 0.0f;
public static final float PROBABILITY_STD = 255.0f;

Fields:-

java
protected Interpreter mtfLite;
private int mImageSizeX;
private int mImageSizeY;
private TensorImage mInputImageBuffer;
private TensorBuffer mOutputProbabilityBuffer;
private TensorProcessor mProbabilityProcessor;
private Bitmap mBitmap;
Uri mImageUri;
private List<String> mLabels;
ImageView mImageView;
Button mButtonPredict;
TextView mPredictText;

Constructors:-

java
public GalleryPredictionActivity()
Constructs a new `GalleryPredictionActivity` object.

Methods:-

onCreate(Bundle savedInstanceState): void`

This method is called when the activity is first created. It initializes the UI components, loads the TensorFlow Lite model, and sets the click listener for the predict button.

Parameters:-

savedInstanceState:

The saved instance state bundle.

loadImage(final Bitmap bitmap): TensorImage`
This method loads a bitmap into a `TensorImage`.

Parameters:-

bitmap: The bitmap to be loaded.

Returns
TensorImage: The loaded `TensorImage`.

loadModelFile(Activity activity):

MappedByteBuffer

This method loads the TensorFlow Lite model.

Parameters
- activity: The current activity.

Returns
- `MappedByteBuffer`: The loaded TensorFlow Lite model.

getPreProcessNormalizeOp(): TensorOperator
This method returns a `TensorOperator` object to normalize the input image.

Returns
TensorOperator: The normalization `TensorOperator`.

getPostProcessNormalizeOp(): TensorOperator
This method returns a `TensorOperator` object to normalize the output probabilities.

Returns
TensorOperator: The normalization `TensorOperator`.

showResult(): void
This method shows the prediction result in the UI.

onActivityResult(int requestCode, int resultCode, Intent data): void
This method is called when the user selects an image from their phone gallery.

Parameters
requestCode: The request code of the activity result.
resultCode: The result code of the activity result.
data: The intent containing the selected image.

static createChooser(Intent target, CharSequence title): Intent
Create a chooser intent to select the X-RAY image.

Parameters:-

target: The intent to be wrapped by the chooser.
title: The title to be shown in the chooser.

Returns
Intent: The created chooser intent.

getPreProcessNormalizeOp(): TensorOperator
This method returns a `TensorOperator` object to normalize the input image.

Returns
TensorOperator: The normalization TensorOperator.

getPostProcessNormalizeOp(): TensorOperator
This method returns a `TensorOperator` object to normalize the output probabilities.

Returns
TensorOperator: The normalization TensorOperator.

TransitionActivity.java:-

The TransitionActivity is an activity class that extends AppCompatActivity. It provides the functionality to transition from the main activity to either the gallery prediction activity or the camera prediction activity.

Fields:-

mGallery: A private Button field that represents the button for transitioning to the gallery prediction activity.

mCamera: A private Button field that represents the button for transitioning to the camera prediction activity.

Methods:-

onCreate(Bundle savedInstanceState): This method is called when the activity is first created. It sets the content view to the activity_transition.xml layout, initializes the mGallery and mCamera buttons, and sets listeners on each button that will start their corresponding prediction activity when clicked.

TEAM_04

Usage:-

To use the TransitionActivity, simply call `startActivity()` with an intent for either the GalleryPredictionActivity or CameraPredictionActivity when the respective button is clicked.

For example

Intent galleryPredictionActivityIntent = new Intent(TransitionActivity.this, GalleryPredictionActivity.class);
startActivity(galleryPredictionActivityIntent);
```

Intent cameraPredictionActivityIntent = new Intent(TransitionActivity.this, CameraPredictionActivity.class);
startActivity(cameraPredictionActivityIntent);
```

It is important to note that both GalleryPredictionActivity and CameraPredictionActivity should be defined in the AndroidManifest.xml file for the application.

For Chat bot we have used an Chat GPT API :-

API Endpoint

The base endpoint for the ChatGPT API is:

https://api.chatgpt.com/v1/

Authentication:-

Requests are authenticated by API keys in the ChatGPT API.
ChatGPT team at support@chatgpt.com to request an API key.

Request Format:-

The ChatGPT API only accepts requests in the form of HTTPS POSTs to the endpoint.
The following fields must be present in a JSON formatted request body:

Example Request:

POST https://api.chatgpt.com/v1/
Content-Type: application/json
Authorization: Bearer API_KEY

```
{
   "text": "hello",
   "model": "gpt2-large",
   "length": 200,
   "top_k": 50,
   "top_p": 0.95
}
```

Example Response:-

json

```
{
   "generated_text": ""Hello! How are you doing today? It's a pleasure to make your
acquaintance.",
   "model": "gpt2-large",
   "request_id": "e0c176a8-b2d2-11eb-8529-0242ac130003"
}
```

Postman:-

Postman, an extremely well-
liked API client, may be used to experiment with and communicate with the ChatGPT API. Here's a sample of a request made over the ChatGPT API using Postman:

Sample request:

```
{
    "model": "gpt-3.5-turbo",
    "messages": [{"role": "user", "content": "Hi, How are you?"}],
    "temperature": 0.7
  }
```

Sample response:


```
{
   "id": "chatcmpl-7AkQXUB0DWCdqgtNJFf53XdhcBzx8",
   "object": "chat.completion",
   "created": 1682796101,
   "model": "gpt-3.5-turbo-0301",
   "usage": {
      "prompt_tokens": 14,
      "completion_tokens": 31,
      "total_tokens": 45
   },
   "choices": [
      {
         "message": {
            "role": "assistant",
            "content": "As an AI language model, I don't have feelings like humans, but I'm
functioning perfectly. Thank you for asking! How may I assist you?"
         },
         "finish_reason": "stop",
         "index": 0
      }
   ]
}
```