VIRTUAL COOKING ASSISTANT NETWORK (V-CAN)

A CAPSTONE PROJECT REPORT

Submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY
IN
COMPUTER SCIENCE ENGINEERING

by

L.SATYAJIT
(18BCD7143)

Under the Guidance of

DR. BKSP KUMAR RAJU ALLURI

SCHOOL OF COMPUTER SCIENCE & ENGINEERING
VIT-AP UNIVERSITY
AMARAVATI- 522237

JANUARY 2022

CERTIFICATE

This is to certify that the Capstone Project work titled "**VIRTUAL COOKING ASSISTANT NETWORK (V-CAN)**" that is being submitted by **L.Satyajit (18BCD7143)** is in partial fulfillment of the requirements for the award of Bachelor of Technology, is a record of bonafide work done under my guidance. The contents of this Project work, in full or in parts, have neither been taken from any other source nor have been submitted to any other Institute or University for award of any degree or diploma and the same is certified.

Dr. BKSP Kumar Raju Alluri

Guide

The thesis is satisfactory / unsatisfactory

Internal Examiner **External Examiner**

Approved by

PROGRAM CHAIR **DEAN**

B. Tech. CSE School Of Computer Science & Engineering

ACKNOWLEDGEMENTS

I am immensely grateful to Dr. BKSP Kumar Raju Alluri for helping me to see through this project. His expertise and constant support was essential in clearing obstacles faced along the way and getting a better understanding on the subject and concepts underlying the project.

I would also like to thank my friends and family who supported me mentally and kept encouraging and motivating me towards the completion of this project.

Finally, My sincere thanks goes to VIT-AP University for providing me with the opportunity to work on this project.

ABSTRACT

A major concern in today's pandemic era is the rapid increase in obesity and other health related chronic diseases based on poor nutrition and diet. Traditional cooking assistant systems as well as calorie tracker applications are tedious to use and lack user friendly features. Additionally the current systems do not work on localized Indian food. In this paper, we present a proof of concept for an efficient cooking assistant system which can automatically detect food and its caloric value from the images captured with a mobile phone camera using deep learning neural networks for image recognition as well as a question answering system based on the transcript of the food recipes. The primary objective of this project is to provide people with an assistance tool for their daily based cooking experience as well as track their calories to promote proper diet which in turn ensures the person is healthy. We also aim to broaden people's knowledge about nutrition, making them more aware of the food they consume on a daily basis. This system will also be easily accessible since it will be deployed in an Android smartphone.

TABLE OF CONTENTS

List of Tables

List of Figures

CHAPTER 1

INTRODUCTION

Lack of proper exercise and improper diet is a major reason for the rise in obesity cases in today's fast paced world. Moreover the influence of the COVID-19 pandemic has been drastic and has led to a massive increase in obesity rates. According to NPR.org, this rise in obesity can be attributed to sedentary lifestyle , job loss and stress. Figure 1 shows the number of diabetic and obese people as well as the total health expenditure on this diseases.

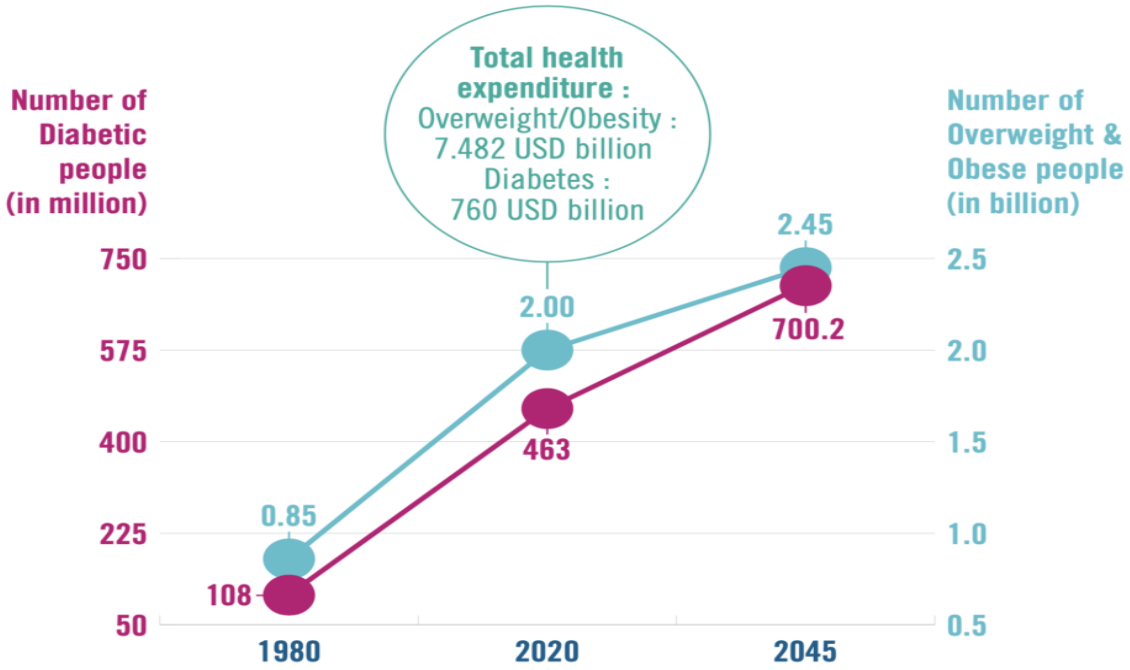


Figure 1 Obesity and Diabetes Cases

The solution for this issue is to make people aware about their nutritional consumption. This can be done with proper diet and keeping a track of calories consumed. However the present systems like “MyFitnessPal” and “Healthify-Me” require manual logging of food making it a tedious task leading to people quit tracking their consumption of calories. Additionally, existing recognition systems do not work on Indian food like Idli, Dosa and Biryani etc. Furthermore, there is a lack of systems which address the doubts people encounter while cooking food.

1.1 Objectives

The following are the objectives of this project:

- To design an efficient and reliable system which can detect food using the images captured from mobile phone cameras.
- To provide calories for the detected food.
- A Question-Answering System based on the transcript of the detected food recipe.
- To deploy this system in an android mobile application to make it easily accessible.

1.2 Background and Literature Survey

A similar project on the same concept was carried out by Research Scholars Arnel B. Ocay, Jane M. Fernandez and Thelma D. Palaoag University of the Cordilleras, Baguio City, Philippines. Their paper [1]“NutriTrack: Android-based food recognition app for nutrition awareness” discussed taking advantage of the recent technological advancements of the mobile phone and using them to address issues, specifically health. Moreover the researcher’s conducted physical experiments to examine the positive impact of using this application.

We also referred to a paper titled [2]“Food Recognition for Dietary Assessment Using Deep Convolutional Neural Networks.” by Professors Stergios Christodoulidis, Marios Anthimopoulos and Stavroula Mougiakakou. This paper talks about a method which involves the implementation of a 6-layer deep convolutional neural network to classify food image patches.

Additionally, We also referred to the paper [3]“Information Retrieval-Based Question Answering System on Foods and Recipes" by Research Scholars Riyanka Manna, Dipankar Das and Alexander Gelbukh. This paper discusses the development of a question answering system on food recipes by using Natural Language Processing (NLP) and Information Retrieval (IR) techniques.

1.3 Organization of the Report

The remaining chapters of the project report are described as follows:

- Chapter 2 contains the proposed system, methodology and software details.
- Chapter 3 briefly describes the datasets involved in the implementation of the project.
- Chapter 4 discusses the results of the implemented mobile applications.
- Chapter 5 concludes the report with suggestions for future work.
- Chapter 6 consists of codes.
- Chapter 7 gives references.

CHAPTER 2

VIRTUAL COOKING ASSISTANT NETWORK (V-CAN)

This Chapter describes the proposed system, working methodology, software and hardware details.

2.1 Proposed System

The following block diagram (figure 2) shows the methodology architecture of this project.

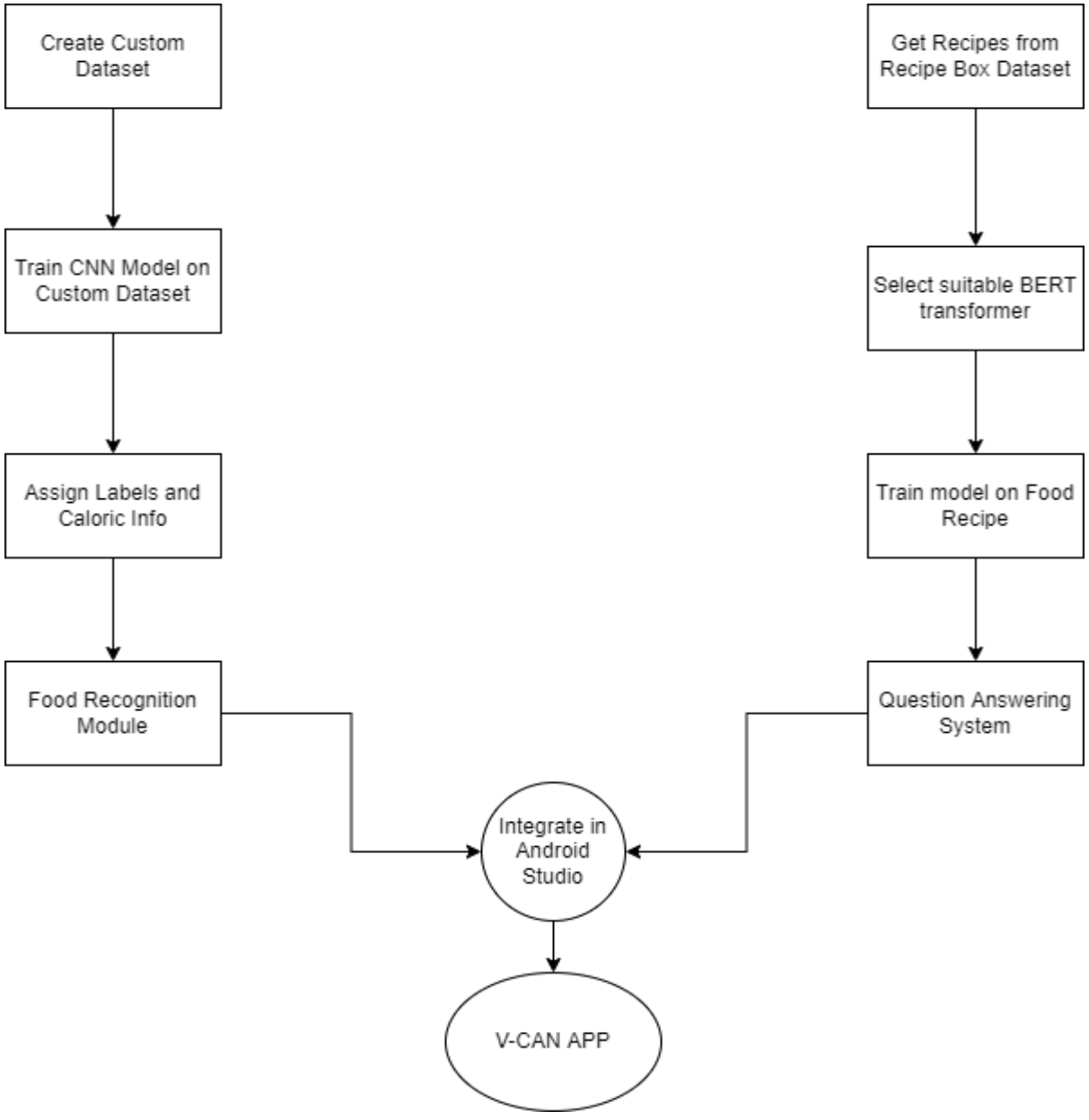


Figure 2 System Block Diagram

2.2 Working Methodology

Data is collected initially from two datasets FOOD-101 and FOOD-20. FOOD-101 consists of 1000 images each of 101 food classes. Similarly, FOOD-20 consists of 100 images each of 20 indian food classes. These two datasets are merged and preprocessed using different techniques such as data augmentation and image cropping. An Inception-V3 CNN model which has been pre-trained on ImageNet dataset is selected for the purpose of training on the merged dataset. Owing to constraints on computational power, We train this model on 10 food classes only of which 5 are Non-Indian food and the other 5 are Indian food. Then we take the saved keras model and convert it into a Tensorflow Lite model so that it can be deployed on a mobile application.

The Recipe Box dataset is used to provide us the food recipes along with the cooking instructions. We then select a MobileBERT model pre-trained on SQUAD (Stanford Question Answering Dataset) 1.1 for the question answering system as it takes less storage and runs much faster than traditional BERT models. We then deploy both these models as mobile applications using Android Studio and the skeleton user interface provided by tensorflow. Table 1 shows the different food classes implemented in this application.

Indian Food	Idly	Biriyani	Chapati	Samosa	Omelette
Non-Indian Food	Pizza	French Fries	Noodles	Sushi	Ice-Crea m

Table 1. Food Classes

2.3 Software Details

Android application and Google Colaboratory are the softwares used in this project

i) Android Application

The Android application is built on the platform Android Studio**.** Android Studio is the development environment officially integrated for Google's Android operating system. It is built on JetBrains' IntelliJ IDEA software and specifically designed for Android development. It allows users to build applications from scratch including the graphical user interface.

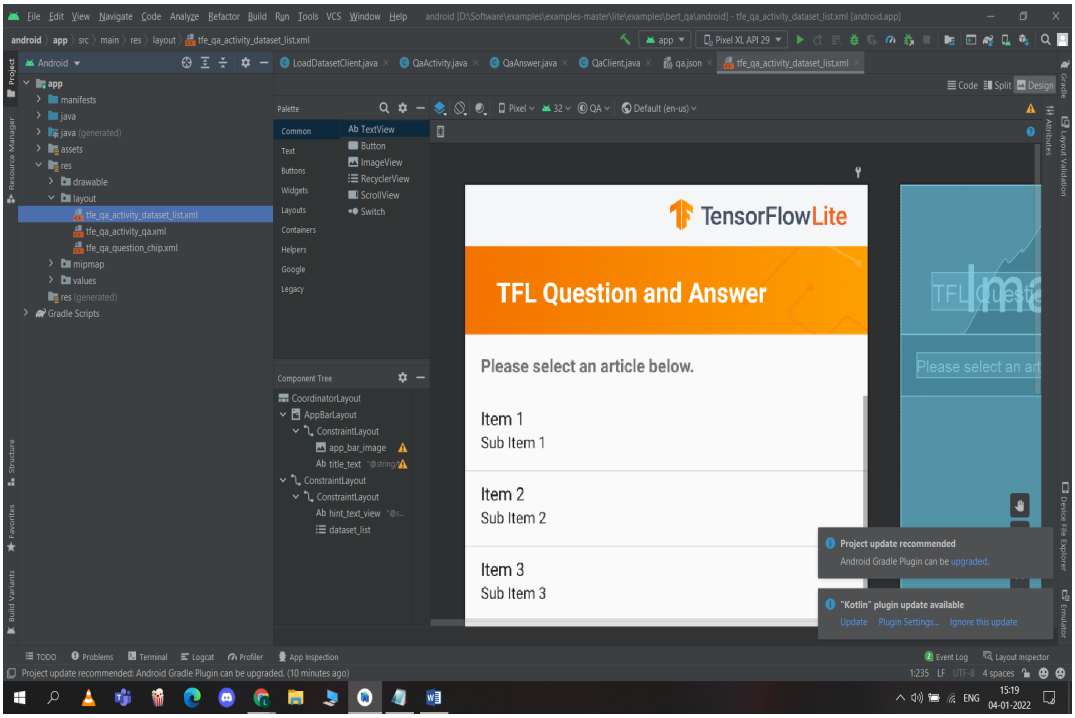


Figure 3 Android Studio Interface

ii) Google Colaboratory

Google Colaboratory or popularly known as Google Colab is a Jupyter environment provided free of cost that runs entirely on cloud servers. This is the platform where we implement the deep learning algorithms and train the models. We also convert the models into Tensorflow Lite models on this Platform.

CHAPTER 3
DATASETS

- **FOOD-101 Dataset**

Food-101 is a dataset consisting of 101 food classes with 750 training and 250 test images per class, making a total of 101,000 images. The test images had already been manually cleaned, while the training set contains some noise. It was introduced in the Paper "Food-101 – Mining Discriminative Components with Random Forests" by Lukas Bossard, Matthieu Guillaumin and Luc Van Gool. The dataset size is 9 GB. A sample of the FOOD-101 dataset is shown in figure 4.

Figure 4 FOOD-101 sample

- **FOOD-20 Dataset**

Food-20 is a dataset consisting of 20 Indian food classes with 70 training and 30 test images per class, making a total of 2000 images. It was uploaded to Kaggle by a user named ‘cdart99’. The dataset size is 460 MB. A sample of the FOOD-101 dataset is shown in figure 5.

Figure 5 FOOD-20 sample

- **Merged Food Dataset**

  Merged food dataset is the dataset created for this project which combines the images of FOOD-101 and FOOD-20. Its total size is almost 10 GB. The google drive link for this dataset is: https://drive.google.com/drive/folders/1MbxN2AUhiABU6jlvocejpy5jUOmfhKY_?usp=sharing

- **Recipe-BOX**

  Recipe box dataset consists of 125,000 recipes scraped from different websites. It is an open source dataset. It includes recipe title, list of ingredients and measurements, instructions for cooking, and a picture of the food.

CHAPTER 4

RESULTS AND DISCUSSIONS

- **Food Recognition CNN Model**

The Food Recognition convolutional neural network model based pretrained Inception V3 model was trained on 1000 images; each of 10 food classes. The model was trained for 20 epochs with a batch size of 30. The validation accuracy was an impressive 94% as shown in figure 6. The accuracy chart per epoch and loss per epoch is shown in figure 7 and figure 8 respectively.

```
Epoch 20/20
161/161 [==============================] - 198s 1s/step - loss: 0.3185 - accuracy: 0.9361 - val_loss: 0.2791 - val_accuracy: 0.9412

Epoch 00020: val_loss improved from 0.28634 to 0.27905, saving model to bestmodel_10class.hdf5
{'biriyani': 0, 'chappati': 1, 'french_fries': 2, 'ice_cream': 3, 'idly': 4, 'noodles': 5, 'omelette': 6, 'pizza': 7, 'samosa': 8, 'sushi': 9}
```

Figure 6 Model Accuracy

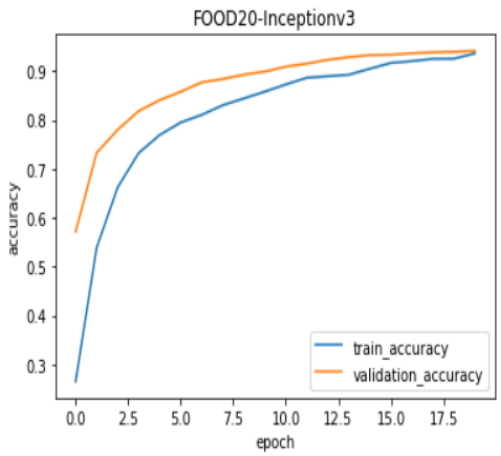


Figure 7 Accuracy plot

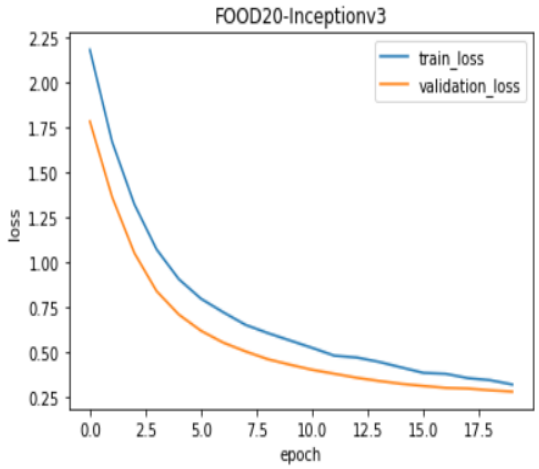


Figure 8 Loss plot

- **Food Recognition Application**

The Food Recognition application based on the Tensorflow Lite model trained on the specified 10 food classes was tested on multiple food images within those 10 classes and detected the food classes with very good accuracy. The test images were collected from the internet as well as images of food captured in personal mobile phones. The final output of the application shows the detected food with the confidence percentage of top-3 classes along with the estimated standard caloric value of the detected food, which is shown in Figure 9.

Figure 9 Food Recognition Application

- **Question-Answering Application**

The application consists of a menu page which shows the 5 distinct options of food recipes, as shown in Figure 10, which on selection of a particular food recipe leads us to the transcript of the food recipe along with the feature to ask personal questions along with some preset questions, as shown in Figure 11. Once the question is typed and selected the MobileBERT model decides the best answer and displays the answer by highlighting the text in yellow along with an audio output of that text using the speakers of the mobile phone as shown in figure 12.

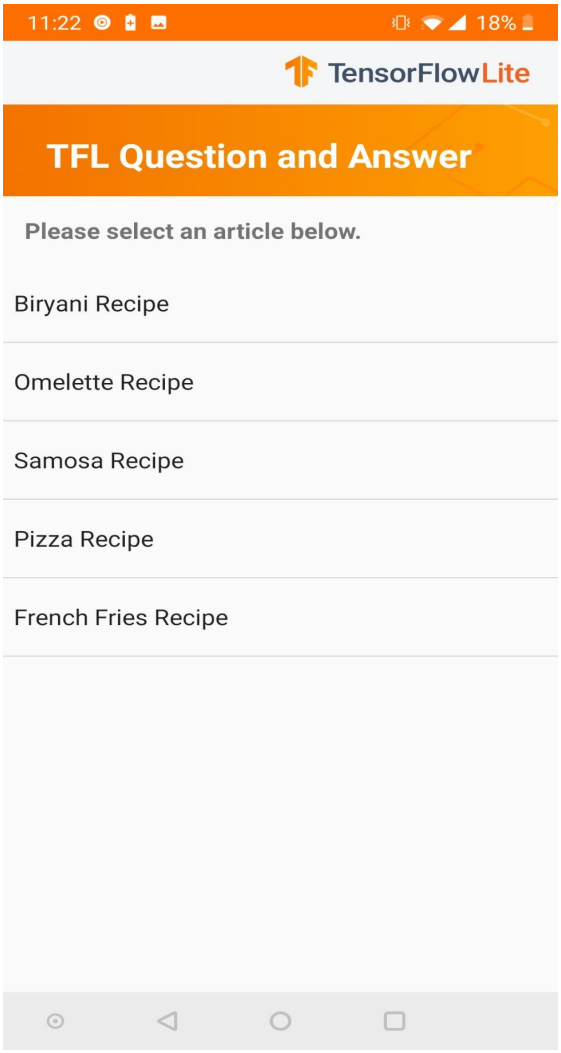


Figure 10 Q&A App Menu page

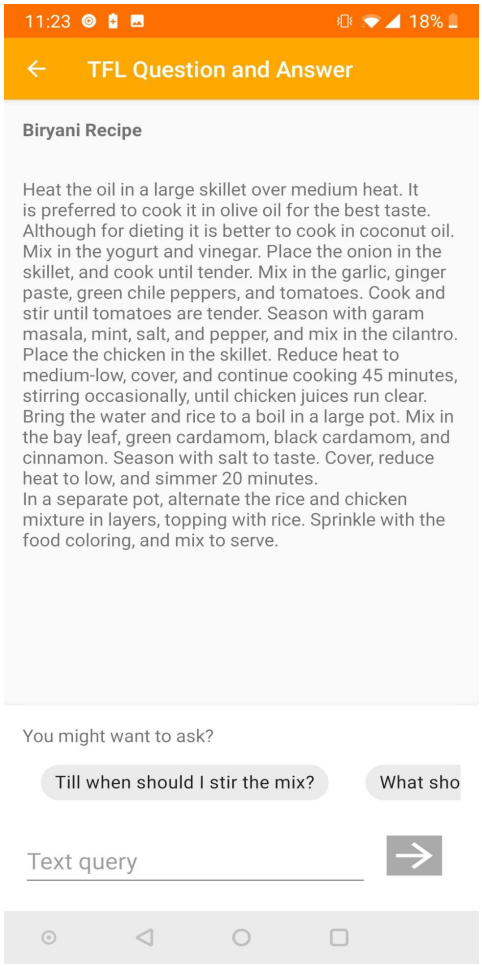


Figure 11 Recipe page

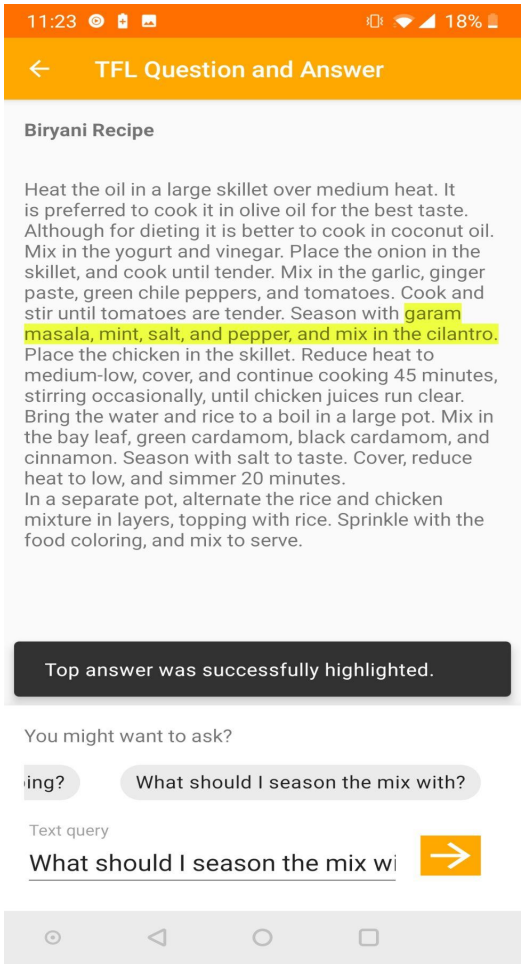


Figure 12 Q&A Output

CHAPTER 5

CONCLUSION AND FUTURE WORK

- To conclude, both of the applications are performing very well and can be treated as a network which can assist the users in cooking as well as calorie tracking experience. Although these applications can be made much better for a much more enriching experience.
- The initial Inception-V3 model had a 94 percent validation accuracy. The model detected all the images when tested manually. However after its conversion to Tensorflow Lite model, though the model became of smaller size, it had reduced validation accuracy when tested on the same image set compared to the initial model. For future upgrades I will look into preserving the validation accuracy of the initial model and minimize the loss in accuracy when converting to a tensorflow lite model. We can also look into the possibility of using an API or cloud based deep learning model for better classification accuracy.
- The TensorFlow model when converted to a Lite model could not be converted to a quantized model. Instead we had to work with a floating point model. The Quantized model is much smaller in size and takes less inference time. The reason why the model could not be converted into a quantized one is still unknown. Significant research in this topic will be done to unearth the cause of this phenomenon.
- Only 10 food classes were trained for this proof of concept due to computational constraints. The dataset created by me consists of 121 food classes, hence with better computational resources we can easily scale this project to a broader dataset.
- We can develop a much more intuitive application with a better user interface as well as combining the features of both the applications into one application.

CHAPTER 6

APPENDIX

Food Detection Code

```
from __future__ import absolute_import, division, print_function
import tensorflow as tf
import tensorflow.keras.backend as K
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing import image
from tensorflow.keras import regularizers
from tensorflow.keras.applications.inception_v3 import InceptionV3
from tensorflow.keras.models import Model
from tensorflow.keras.layers import Dense, Dropout
from tensorflow.keras.layers import GlobalAveragePooling2D
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.callbacks import ModelCheckpoint, CSVLogger
from tensorflow.keras.optimizers import SGD
from tensorflow.keras.regularizers import l2
from tensorflow import keras
from tensorflow.keras import models
from tensorflow.keras.applications.inception_v3 import preprocess_input
import cv2
import os
import random
import collections
from collections import defaultdict
from shutil import copy
from shutil import copytree, rmtree
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.image as img
%matplotlib inline
# Check TF version and whether GPU is enabled
print(tf.__version__)
print(tf.test.gpu_device_name())
```

```
# Clone tensorflow/examples repo which has images to evaluate trained model
!git clone https://github.com/tensorflow/examples.git
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
import os
#os.chdir("/content/drive/MyDrive")
!unrar x "/content/drive/MyDrive/food20dataset.rar" "food20dataset"
print("Total number of samples in train folder")
train_files = sum([len(files) for i, j, files in
os.walk("food20dataset/train_set")])
print(train_files)
print("Total number of samples in test folder")
test_files = sum([len(files) for i, j, files in
os.walk("food20dataset/test_set")])
print(test_files)
# Helper method to create train_mini and test_mini data samples
def dataset_mini(food_list, src, dest):
  if os.path.exists(dest):
    rmtree(dest) # removing dataset_mini(if it already exists) folders so
that we will have only the classes that we want
  os.makedirs(dest)
  for food_item in food_list :
    print("Copying images into",food_item)
    copytree(os.path.join(src,food_item), os.path.join(dest,food_item))
# picking 3 food items and generating separate data folders for the same
food_list =
['biriyani','chappati','french_fries','ice_cream','idly','noodles','omelett
e','pizza','samosa','sushi']
src_train = 'food20dataset/train_set'
dest_train = 'food20dataset/train_mini'
src_test = 'food20dataset/test_set'
dest_test = 'food20dataset/test_mini'
print("Creating train data folder with new classes")
dataset_mini(food_list, src_train, dest_train)
print("Total number of samples in train folder")
train_files = sum([len(files) for i, j, files in
os.walk("food20dataset/train_mini")])
print(train_files)
print("Creating test data folder with new classes")
```

```
dataset_mini(food_list, src_test, dest_test)
print("Total number of samples in test folder")
test_files = sum([len(files) for i, j, files in
os.walk("food20dataset/test_mini")])
print(test_files)
def train_model(n_classes,num_epochs,
nb_train_samples,nb_validation_samples):
  K.clear_session()

  img_width, img_height = 299, 299
  train_data_dir = 'food20dataset/train_mini'
  validation_data_dir = 'food20dataset/test_mini'
  batch_size = 30
  bestmodel_path = 'bestmodel_'+str(n_classes)+'class.hdf5'
  trainedmodel_path = 'trainedmodel_'+str(n_classes)+'class.hdf5'
  history_path = 'history_'+str(n_classes)+'.log'

  train_datagen = ImageDataGenerator(
      preprocessing_function=preprocess_input,
      shear_range=0.2,
      zoom_range=0.2,
      horizontal_flip=True)

  test_datagen =
ImageDataGenerator(preprocessing_function=preprocess_input)

  train_generator = train_datagen.flow_from_directory(
      train_data_dir,
      target_size=(img_height, img_width),
      batch_size=batch_size,
      class_mode='categorical')

  validation_generator = test_datagen.flow_from_directory(
      validation_data_dir,
      target_size=(img_height, img_width),
      batch_size=batch_size,
      class_mode='categorical')
```

```
  inception = InceptionV3(weights='imagenet', include_top=False)
  x = inception.output
  x = GlobalAveragePooling2D()(x)
  x = Dense(128,activation='relu')(x)
  x = Dropout(0.2)(x)

  predictions = Dense(n_classes,kernel_regularizer=regularizers.l2(0.005),
activation='softmax')(x)

  model = Model(inputs=inception.input, outputs=predictions)
  model.compile(optimizer=SGD(lr=0.0001, momentum=0.9),
loss='categorical_crossentropy', metrics=['accuracy'])
  checkpoint = ModelCheckpoint(filepath=bestmodel_path, verbose=1,
save_best_only=True)
  csv_logger = CSVLogger(history_path)

  history = model.fit_generator(train_generator,
                      steps_per_epoch = nb_train_samples // batch_size,
                      validation_data=validation_generator,
                      validation_steps=nb_validation_samples // batch_size,
                      epochs=num_epochs,
                      verbose=1,
                      callbacks=[csv_logger, checkpoint])

  model.save(trainedmodel_path)
  class_map = train_generator.class_indices
  return history, class_map
# Train the model with data from 3 classes
n_classes = 10
epochs = 20
nb_train_samples = train_files
nb_validation_samples = test_files

history, class_map_3 = train_model(n_classes,epochs,
nb_train_samples,nb_validation_samples)
print(class_map_3)
# Visualize the accuracy and loss plots
def plot_accuracy(history,title):
    plt.title(title)
```

```
    plt.plot(history.history['accuracy']) # change acc to accuracy if
testing TF 2.0
    plt.plot(history.history['val_accuracy']) # change val_accuracy if
testing TF 2.0
    plt.ylabel('accuracy')
    plt.xlabel('epoch')
    plt.legend(['train_accuracy', 'validation_accuracy'], loc='best')
    plt.show()


def plot_loss(history,title):
    plt.title(title)
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.ylabel('loss')
    plt.xlabel('epoch')
    plt.legend(['train_loss', 'validation_loss'], loc='best')
    plt.show()


plot_accuracy(history,'FOOD20-Inceptionv3')
plot_loss(history,'FOOD20-Inceptionv3')
%%time
# Loading the best saved model to make predictions

K.clear_session()
model_best =
load_model('/content/drive/MyDrive/bestmodel_10class.hdf5',compile = False)
model_best = load_model('/content/keras_model.h5',compile = False)
##food_list = ['biriyani','pizza','ice_cream','idly','samosa']
def predict_class(model, images, show = True):
  for img in images:
    img = image.load_img(img, target_size=( 224, 224, 3))
    img = image.img_to_array(img)
    img = np.expand_dims(img, axis=0)
    img = preprocess_input(img)

    pred = model.predict(img)
    index = np.argmax(pred)
```

```
    pred_value = food_list[index]
    print(pred)
    if show:
        plt.imshow(img[0])
        plt.axis('off')
        plt.title(pred_value)
        plt.show()
    return pred_value
# Make a list of downloaded images and test the trained model
images = []
images.append('1.jpg')
images.append('2.jpg')
images.append('3.jpg')
images.append('Pizza.jpg')
images.append('4.jpg')
images.append('5.jpg')
images.append('6.jpg')
images.append('Omelette.jpg')
images.append('7.jpg')
images.append('8.jpg')
images.append('9.jpg')

images.append('10.jpg')
images.append('11.jpg')
images.append('samosa.jpg')

images.append('12.jpg')
images.append('13.jpg')
images.append('14.jpg')
images.append('15.jpg')
images.append('16.jpg')
predict_class(model_best, images, True)
```

Food Detection Android Studio Code

Build Gradle

```
apply plugin: 'com.android.application'

android {
   compileSdkVersion 32
   defaultConfig {
       applicationId "org.tensorflow.lite.examples.classification"
       minSdkVersion 21
       targetSdkVersion 32
       versionCode 1
       versionName "1.0"
   }
   buildTypes {
       release {
           minifyEnabled false
           proguardFiles getDefaultProguardFile('proguard-android.txt'),
'proguard-rules.pro'
       }
   }
   aaptOptions {
       noCompress "tflite"
   }
   compileOptions {
       sourceCompatibility = '1.8'
       targetCompatibility = '1.8'
   }
   lintOptions {
       abortOnError false
   }
}

// Download default models; if you wish to use your own models then
// place them in the "assets" directory and comment out this line.
apply from:'download.gradle'

dependencies {
   implementation fileTree(dir: 'libs', include: ['*.jar'])
   implementation 'androidx.appcompat:appcompat:1.0.0'
   implementation 'androidx.coordinatorlayout:coordinatorlayout:1.0.0'
   implementation 'com.google.android.material:material:1.0.0'

   // Build off of nightly TensorFlow Lite
   implementation('org.tensorflow:tensorflow-lite:0.0.0-nightly') {
changing = true }
   implementation('org.tensorflow:tensorflow-lite-gpu:0.0.0-nightly') {
changing = true }
   implementation('org.tensorflow:tensorflow-lite-support:0.0.0-nightly') {
changing = true }
   // Use local TensorFlow library
```

```
   // implementation 'org.tensorflow:tensorflow-lite-local:0.0.0'

   androidTestImplementation 'androidx.test.ext:junit:1.1.1'
   androidTestImplementation 'com.android.support.test:rules:1.0.2'
   androidTestImplementation 'com.google.truth:truth:1.0.1'
}
```

Classifier Activity

```
package org.tensorflow.lite.examples.classification;

import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Typeface;
import android.media.ImageReader.OnImageAvailableListener;
import android.os.SystemClock;
import android.util.Size;
import android.util.TypedValue;
import android.widget.Toast;
import java.io.IOException;
import java.util.List;
import org.tensorflow.lite.examples.classification.env.BorderedText;
import org.tensorflow.lite.examples.classification.env.Logger;
import org.tensorflow.lite.examples.classification.tflite.Classifier;
import
org.tensorflow.lite.examples.classification.tflite.Classifier.Device;
import org.tensorflow.lite.examples.classification.tflite.Classifier.Model;

public class ClassifierActivity extends CameraActivity implements
OnImageAvailableListener {
 private static final Logger LOGGER = new Logger();
 private static final Size DESIRED_PREVIEW_SIZE = new Size(640, 480);
 private static final float TEXT_SIZE_DIP = 10;
 private Bitmap rgbFrameBitmap = null;
 private long lastProcessingTimeMs;
 private Integer sensorOrientation;
 private Classifier classifier;
 private BorderedText borderedText;
 /** Input image size of the model along x axis. */
 private int imageSizeX;
 /** Input image size of the model along y axis. */
 private int imageSizeY;

 @Override
 protected int getLayoutId() {
   return R.layout.tfe_ic_camera_connection_fragment;
 }

 @Override
 protected Size getDesiredPreviewFrameSize() {
   return DESIRED_PREVIEW_SIZE;
```

```
 }

 @Override
 public void onPreviewSizeChosen(final Size size, final int rotation) {
   final float textSizePx =
       TypedValue.applyDimension(
           TypedValue.COMPLEX_UNIT_DIP, TEXT_SIZE_DIP,
getResources().getDisplayMetrics());
   borderedText = new BorderedText(textSizePx);
   borderedText.setTypeface(Typeface.MONOSPACE);

   recreateClassifier(getModel(), getDevice(), getNumThreads());
   if (classifier == null) {
     LOGGER.e("No classifier on preview!");
     return;
   }

   previewWidth = size.getWidth();
   previewHeight = size.getHeight();

   sensorOrientation = rotation - getScreenOrientation();
   LOGGER.i("Camera orientation relative to screen canvas: %d",
sensorOrientation);

   LOGGER.i("Initializing at size %dx%d", previewWidth, previewHeight);
   rgbFrameBitmap = Bitmap.createBitmap(previewWidth, previewHeight,
Config.ARGB_8888);
 }

 @Override
 protected void processImage() {
   rgbFrameBitmap.setPixels(getRgbBytes(), 0, previewWidth, 0, 0,
previewWidth, previewHeight);
   final int cropSize = Math.min(previewWidth, previewHeight);

   runInBackground(
       new Runnable() {
         @Override
         public void run() {
           if (classifier != null) {
             final long startTime = SystemClock.uptimeMillis();
             final List<Classifier.Recognition> results =
                 classifier.recognizeImage(rgbFrameBitmap,
sensorOrientation);
             lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;
             LOGGER.v("Detect: %s", results);

             runOnUiThread(
                 new Runnable() {
                   @Override
                   public void run() {
                     showResultsInBottomSheet(results);
                     showFrameInfo(previewWidth + "x" + previewHeight);
```

```
                    showCropInfo(imageSizeX + "x" + imageSizeY);
                    showCameraResolution(cropSize + "x" + cropSize);
                    showRotationInfo(String.valueOf(sensorOrientation));
                    showInference(lastProcessingTimeMs + "ms");
                  }
                });
          }
          readyForNextImage();
        }
      });
 }

 @Override
 protected void onInferenceConfigurationChanged() {
   if (rgbFrameBitmap == null) {
     // Defer creation until we're getting camera frames.
     return;
   }
   final Device device = getDevice();
   final Model model = getModel();
   final int numThreads = getNumThreads();
   runInBackground(() -> recreateClassifier(model, device, numThreads));
 }

 private void recreateClassifier(Model model, Device device, int
numThreads) {
   if (classifier != null) {
     LOGGER.d("Closing classifier.");
     classifier.close();
     classifier = null;
   }
   if (device == Device.GPU
       && (model == Model.QUANTIZED_MOBILENET || model ==
Model.QUANTIZED_EFFICIENTNET)) {
     LOGGER.d("Not creating classifier: GPU doesn't support quantized
models.");
     runOnUiThread(
         () -> {
           Toast.makeText(this, R.string.tfe_ic_gpu_quant_error,
Toast.LENGTH_LONG).show();
         });
     return;
   }
   try {
     LOGGER.d(
         "Creating classifier (model=%s, device=%s, numThreads=%d)", model,
device, numThreads);
     classifier = Classifier.create(this, model, device, numThreads);
   } catch (IOException e) {
     LOGGER.e(e, "Failed to create classifier.");
   }

   // Updates the input image size.
```

```
    imageSizeX = classifier.getImageSizeX();
    imageSizeY = classifier.getImageSizeY();
  }
}
```

Camera Activity

```
package org.tensorflow.lite.examples.classification;

import android.Manifest;
import android.app.Fragment;
import android.content.Context;
import android.content.pm.PackageManager;
import android.hardware.Camera;
import android.hardware.camera2.CameraAccessException;
import android.hardware.camera2.CameraCharacteristics;
import android.hardware.camera2.CameraManager;
import android.hardware.camera2.params.StreamConfigurationMap;
import android.media.Image;
import android.media.Image.Plane;
import android.media.ImageReader;
import android.media.ImageReader.OnImageAvailableListener;
import android.os.Build;
import android.os.Bundle;
import android.os.Handler;
import android.os.HandlerThread;
import android.os.Trace;
import androidx.annotation.NonNull;
import androidx.annotation.UiThread;
import androidx.appcompat.app.AppCompatActivity;
import android.util.Size;
import android.view.Surface;
import android.view.View;
import android.view.ViewTreeObserver;
import android.view.WindowManager;
import android.widget.AdapterView;
import android.widget.ImageView;
import android.widget.LinearLayout;
import android.widget.Spinner;
import android.widget.TextView;
import android.widget.Toast;
import com.google.android.material.bottomsheet.BottomSheetBehavior;
import java.nio.ByteBuffer;
import java.util.List;
import org.tensorflow.lite.examples.classification.env.ImageUtils;
import org.tensorflow.lite.examples.classification.env.Logger;
import
org.tensorflow.lite.examples.classification.tflite.Classifier.Device;
import org.tensorflow.lite.examples.classification.tflite.Classifier.Model;
import
org.tensorflow.lite.examples.classification.tflite.Classifier.Recognition;
```

```
public abstract class CameraActivity extends AppCompatActivity
   implements OnImageAvailableListener,
       Camera.PreviewCallback,
       View.OnClickListener,
       AdapterView.OnItemSelectedListener {
 private static final Logger LOGGER = new Logger();

 private static final int PERMISSIONS_REQUEST = 1;

 private static final String PERMISSION_CAMERA =
Manifest.permission.CAMERA;
 protected int previewWidth = 0;
 protected int previewHeight = 0;
 private Handler handler;
 private HandlerThread handlerThread;
 private boolean useCamera2API;
 private boolean isProcessingFrame = false;
 private byte[][] yuvBytes = new byte[3][];
 private int[] rgbBytes = null;
 private int yRowStride;
 private Runnable postInferenceCallback;
 private Runnable imageConverter;
 private LinearLayout bottomSheetLayout;
 private LinearLayout gestureLayout;
 private BottomSheetBehavior<LinearLayout> sheetBehavior;
 protected TextView recognitionTextView,
     recognition1TextView,
     recognition2TextView,
     recognitionValueTextView,
     recognition1ValueTextView,
     recognition2ValueTextView;
 protected TextView frameValueTextView,
     cropValueTextView,
     cameraResolutionTextView,
     rotationTextView,
     inferenceTimeTextView;
 protected ImageView bottomSheetArrowImageView;
 private ImageView plusImageView, minusImageView;
 private Spinner modelSpinner;
 private Spinner deviceSpinner;
 private TextView threadsTextView;

 private Model model = Model.FLOAT_EFFICIENTNET;
 private Device device = Device.CPU;
 private int numThreads = -1;

 @Override
 protected void onCreate(final Bundle savedInstanceState) {
   LOGGER.d("onCreate " + this);
   super.onCreate(null);
   getWindow().addFlags(WindowManager.LayoutParams.FLAG_KEEP_SCREEN_ON);

   setContentView(R.layout.tfe_ic_activity_camera);
```

```
   if (hasPermission()) {
     setFragment();
   } else {
     requestPermission();
   }

   threadsTextView = findViewById(R.id.threads);
   plusImageView = findViewById(R.id.plus);
   minusImageView = findViewById(R.id.minus);
   modelSpinner = findViewById(R.id.model_spinner);
   deviceSpinner = findViewById(R.id.device_spinner);
   bottomSheetLayout = findViewById(R.id.bottom_sheet_layout);
   gestureLayout = findViewById(R.id.gesture_layout);
   sheetBehavior = BottomSheetBehavior.from(bottomSheetLayout);
   bottomSheetArrowImageView = findViewById(R.id.bottom_sheet_arrow);

   ViewTreeObserver vto = gestureLayout.getViewTreeObserver();
   vto.addOnGlobalLayoutListener(
       new ViewTreeObserver.OnGlobalLayoutListener() {
         @Override
         public void onGlobalLayout() {
           if (Build.VERSION.SDK_INT < Build.VERSION_CODES.JELLY_BEAN) {

gestureLayout.getViewTreeObserver().removeGlobalOnLayoutListener(this);
           } else {

gestureLayout.getViewTreeObserver().removeOnGlobalLayoutListener(this);
           }
           //                int width =
bottomSheetLayout.getMeasuredWidth();
           int height = gestureLayout.getMeasuredHeight();

           sheetBehavior.setPeekHeight(height);
         }
       });
   sheetBehavior.setHideable(false);

   sheetBehavior.setBottomSheetCallback(
       new BottomSheetBehavior.BottomSheetCallback() {
         @Override
         public void onStateChanged(@NonNull View bottomSheet, int
newState) {
           switch (newState) {
             case BottomSheetBehavior.STATE_HIDDEN:
               break;
             case BottomSheetBehavior.STATE_EXPANDED:
               {

bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_down);
               }
               break;
             case BottomSheetBehavior.STATE_COLLAPSED:
```

```
                {

bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_up);
                }
                break;
              case BottomSheetBehavior.STATE_DRAGGING:
                break;
              case BottomSheetBehavior.STATE_SETTLING:

bottomSheetArrowImageView.setImageResource(R.drawable.icn_chevron_up);
                break;
            }
          }

          @Override
          public void onSlide(@NonNull View bottomSheet, float slideOffset)
{}
        });

    recognitionTextView = findViewById(R.id.detected_item);
    recognitionValueTextView = findViewById(R.id.detected_item_value);
    recognition1TextView = findViewById(R.id.detected_item1);
    recognition1ValueTextView = findViewById(R.id.detected_item1_value);
    recognition2TextView = findViewById(R.id.detected_item2);
    recognition2ValueTextView = findViewById(R.id.detected_item2_value);

    frameValueTextView = findViewById(R.id.frame_info);
    cropValueTextView = findViewById(R.id.crop_info);
    cameraResolutionTextView = findViewById(R.id.view_info);
    rotationTextView = findViewById(R.id.rotation_info);
    inferenceTimeTextView = findViewById(R.id.inference_info);

    modelSpinner.setOnItemSelectedListener(this);
    deviceSpinner.setOnItemSelectedListener(this);

    plusImageView.setOnClickListener(this);
    minusImageView.setOnClickListener(this);

    model =
Model.valueOf(modelSpinner.getSelectedItem().toString().toUpperCase());
    device = Device.valueOf(deviceSpinner.getSelectedItem().toString());
    numThreads =
Integer.parseInt(threadsTextView.getText().toString().trim());
  }

  protected int[] getRgbBytes() {
    imageConverter.run();
    return rgbBytes;
  }

  protected int getLuminanceStride() {
    return yRowStride;
  }
```

```
 protected byte[] getLuminance() {
   return yuvBytes[0];
 }

 /** Callback for android.hardware.Camera API */
 @Override
 public void onPreviewFrame(final byte[] bytes, final Camera camera) {
   if (isProcessingFrame) {
     LOGGER.w("Dropping frame!");
     return;
   }

   try {
     // Initialize the storage bitmaps once when the resolution is known.
     if (rgbBytes == null) {
       Camera.Size previewSize = camera.getParameters().getPreviewSize();
       previewHeight = previewSize.height;
       previewWidth = previewSize.width;
       rgbBytes = new int[previewWidth * previewHeight];
       onPreviewSizeChosen(new Size(previewSize.width, previewSize.height),
90);
     }
   } catch (final Exception e) {
     LOGGER.e(e, "Exception!");
     return;
   }

   isProcessingFrame = true;
   yuvBytes[0] = bytes;
   yRowStride = previewWidth;

   imageConverter =
       new Runnable() {
         @Override
         public void run() {
           ImageUtils.convertYUV420SPToARGB8888(bytes, previewWidth,
previewHeight, rgbBytes);
         }
       };

   postInferenceCallback =
       new Runnable() {
         @Override
         public void run() {
           camera.addCallbackBuffer(bytes);
           isProcessingFrame = false;
         }
       };
   processImage();
 }

 /** Callback for Camera2 API */
```

```
@Override
public void onImageAvailable(final ImageReader reader) {
  // We need wait until we have some size from onPreviewSizeChosen
  if (previewWidth == 0 || previewHeight == 0) {
    return;
  }
  if (rgbBytes == null) {
    rgbBytes = new int[previewWidth * previewHeight];
  }
  try {
    final Image image = reader.acquireLatestImage();

    if (image == null) {
      return;
    }

    if (isProcessingFrame) {
      image.close();
      return;
    }
    isProcessingFrame = true;
    Trace.beginSection("imageAvailable");
    final Plane[] planes = image.getPlanes();
    fillBytes(planes, yuvBytes);
    yRowStride = planes[0].getRowStride();
    final int uvRowStride = planes[1].getRowStride();
    final int uvPixelStride = planes[1].getPixelStride();

    imageConverter =
        new Runnable() {
          @Override
          public void run() {
            ImageUtils.convertYUV420ToARGB8888(
                yuvBytes[0],
                yuvBytes[1],
                yuvBytes[2],
                previewWidth,
                previewHeight,
                yRowStride,
                uvRowStride,
                uvPixelStride,
                rgbBytes);
          }
        };

    postInferenceCallback =
        new Runnable() {
          @Override
          public void run() {
            image.close();
            isProcessingFrame = false;
          }
        };
```

```
    processImage();
  } catch (final Exception e) {
    LOGGER.e(e, "Exception!");
    Trace.endSection();
    return;
  }
  Trace.endSection();
}

@Override
public synchronized void onStart() {
  LOGGER.d("onStart " + this);
  super.onStart();
}

@Override
public synchronized void onResume() {
  LOGGER.d("onResume " + this);
  super.onResume();

  handlerThread = new HandlerThread("inference");
  handlerThread.start();
  handler = new Handler(handlerThread.getLooper());
}

@Override
public synchronized void onPause() {
  LOGGER.d("onPause " + this);

  handlerThread.quitSafely();
  try {
    handlerThread.join();
    handlerThread = null;
    handler = null;
  } catch (final InterruptedException e) {
    LOGGER.e(e, "Exception!");
  }

  super.onPause();
}

@Override
public synchronized void onStop() {
  LOGGER.d("onStop " + this);
  super.onStop();
}

@Override
public synchronized void onDestroy() {
  LOGGER.d("onDestroy " + this);
  super.onDestroy();
}
```

```
 protected synchronized void runInBackground(final Runnable r) {
   if (handler != null) {
     handler.post(r);
   }
 }

 @Override
 public void onRequestPermissionsResult(
     final int requestCode, final String[] permissions, final int[]
grantResults) {
   super.onRequestPermissionsResult(requestCode, permissions,
grantResults);
   if (requestCode == PERMISSIONS_REQUEST) {
     if (allPermissionsGranted(grantResults)) {
       setFragment();
     } else {
       requestPermission();
     }
   }
 }

 private static boolean allPermissionsGranted(final int[] grantResults) {
   for (int result : grantResults) {
     if (result != PackageManager.PERMISSION_GRANTED) {
       return false;
     }
   }
   return true;
 }

 private boolean hasPermission() {
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
     return checkSelfPermission(PERMISSION_CAMERA) ==
PackageManager.PERMISSION_GRANTED;
   } else {
     return true;
   }
 }

 private void requestPermission() {
   if (Build.VERSION.SDK_INT >= Build.VERSION_CODES.M) {
     if (shouldShowRequestPermissionRationale(PERMISSION_CAMERA)) {
       Toast.makeText(
               CameraActivity.this,
               "Camera permission is required for this demo",
               Toast.LENGTH_LONG)
           .show();
     }
     requestPermissions(new String[] {PERMISSION_CAMERA},
PERMISSIONS_REQUEST);
   }
 }
```

```
 // Returns true if the device supports the required hardware level, or
better.
 private boolean isHardwareLevelSupported(
     CameraCharacteristics characteristics, int requiredLevel) {
   int deviceLevel =
characteristics.get(CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL);
   if (deviceLevel ==
CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_LEGACY) {
     return requiredLevel == deviceLevel;
   }
   // deviceLevel is not LEGACY, can use numerical sort
   return requiredLevel <= deviceLevel;
 }

 private String chooseCamera() {
   final CameraManager manager = (CameraManager)
getSystemService(Context.CAMERA_SERVICE);
   try {
     for (final String cameraId : manager.getCameraIdList()) {
       final CameraCharacteristics characteristics =
manager.getCameraCharacteristics(cameraId);

       // We don't use a front facing camera in this sample.
       final Integer facing =
characteristics.get(CameraCharacteristics.LENS_FACING);
       if (facing != null && facing ==
CameraCharacteristics.LENS_FACING_FRONT) {
         continue;
       }

       final StreamConfigurationMap map =

characteristics.get(CameraCharacteristics.SCALER_STREAM_CONFIGURATION_MAP);

       if (map == null) {
         continue;
       }

       // Fallback to camera1 API for internal cameras that don't have full
support.
       // This should help with legacy situations where using the camera2
API causes
       // distorted or otherwise broken previews.
       useCamera2API =
           (facing == CameraCharacteristics.LENS_FACING_EXTERNAL)
               || isHardwareLevelSupported(
                   characteristics,
CameraCharacteristics.INFO_SUPPORTED_HARDWARE_LEVEL_FULL);
       LOGGER.i("Camera API lv2?: %s", useCamera2API);
       return cameraId;
     }
   } catch (CameraAccessException e) {
```

```
      LOGGER.e(e, "Not allowed to access camera");
    }

    return null;
  }

  protected void setFragment() {
    String cameraId = chooseCamera();

    Fragment fragment;
    if (useCamera2API) {
      CameraConnectionFragment camera2Fragment =
          CameraConnectionFragment.newInstance(
              new CameraConnectionFragment.ConnectionCallback() {
                @Override
                public void onPreviewSizeChosen(final Size size, final int
rotation) {
                  previewHeight = size.getHeight();
                  previewWidth = size.getWidth();
                  CameraActivity.this.onPreviewSizeChosen(size, rotation);
                }
              },
              this,
              getLayoutId(),
              getDesiredPreviewFrameSize());

      camera2Fragment.setCamera(cameraId);
      fragment = camera2Fragment;
    } else {
      fragment =
          new LegacyCameraConnectionFragment(this, getLayoutId(),
getDesiredPreviewFrameSize());
    }

    getFragmentManager().beginTransaction().replace(R.id.container,
fragment).commit();
  }

  protected void fillBytes(final Plane[] planes, final byte[][] yuvBytes) {
    // Because of the variable row stride it's not possible to know in
    // advance the actual necessary dimensions of the yuv planes.
    for (int i = 0; i < planes.length; ++i) {
      final ByteBuffer buffer = planes[i].getBuffer();
      if (yuvBytes[i] == null) {
        LOGGER.d("Initializing buffer %d at size %d", i, buffer.capacity());
        yuvBytes[i] = new byte[buffer.capacity()];
      }
      buffer.get(yuvBytes[i]);
    }
  }

  protected void readyForNextImage() {
    if (postInferenceCallback != null) {
```

```
      postInferenceCallback.run();
    }
  }

  protected int getScreenOrientation() {
    switch (getWindowManager().getDefaultDisplay().getRotation()) {
      case Surface.ROTATION_270:
        return 270;
      case Surface.ROTATION_180:
        return 180;
      case Surface.ROTATION_90:
        return 90;
      default:
        return 0;
    }
  }

  @UiThread
  protected void showResultsInBottomSheet(List<Recognition> results) {
    if (results != null && results.size() >= 3) {
      Recognition recognition = results.get(0);
      if (recognition != null) {
        if (recognition.getTitle() != null)
recognitionTextView.setText(recognition.getTitle());
        if (recognition.getConfidence() != null)
          recognitionValueTextView.setText(
              String.format("%.2f", (100 * recognition.getConfidence())) +
"%");
      }

      Recognition recognition1 = results.get(1);
      if (recognition1 != null) {
        if (recognition1.getTitle() != null)
recognition1TextView.setText(recognition1.getTitle());
        if (recognition1.getConfidence() != null)
          recognition1ValueTextView.setText(
              String.format("%.2f", (100 * recognition1.getConfidence())) +
"%");
      }

      Recognition recognition2 = results.get(2);
      if (recognition2 != null) {
        if (recognition2.getTitle() != null)
recognition2TextView.setText(recognition2.getTitle());
        if (recognition2.getConfidence() != null)
          recognition2ValueTextView.setText(
              String.format("%.2f", (100 * recognition2.getConfidence())) +
"%");
      }
    }
  }

  protected void showFrameInfo(String frameInfo) {
```

```
    frameValueTextView.setText(frameInfo);
  }

  protected void showCropInfo(String cropInfo) {
    cropValueTextView.setText(cropInfo);
  }

  protected void showCameraResolution(String cameraInfo) {
    cameraResolutionTextView.setText(cameraInfo);
  }

  protected void showRotationInfo(String rotation) {
    rotationTextView.setText(rotation);
  }

  protected void showInference(String inferenceTime) {
    inferenceTimeTextView.setText(inferenceTime);
  }

  protected Model getModel() {
    return model;
  }

  private void setModel(Model model) {
    if (this.model != model) {
      LOGGER.d("Updating  model: " + model);
      this.model = model;
      onInferenceConfigurationChanged();
    }
  }

  protected Device getDevice() {
    return device;
  }

  private void setDevice(Device device) {
    if (this.device != device) {
      LOGGER.d("Updating  device: " + device);
      this.device = device;
      final boolean threadsEnabled = device == Device.CPU;
      plusImageView.setEnabled(threadsEnabled);
      minusImageView.setEnabled(threadsEnabled);
      threadsTextView.setText(threadsEnabled ? String.valueOf(numThreads) :
"N/A");
      onInferenceConfigurationChanged();
    }
  }

  protected int getNumThreads() {
    return numThreads;
  }

  private void setNumThreads(int numThreads) {
```

```
    if (this.numThreads != numThreads) {
      LOGGER.d("Updating  numThreads: " + numThreads);
      this.numThreads = numThreads;
      onInferenceConfigurationChanged();
    }
  }

  protected abstract void processImage();

  protected abstract void onPreviewSizeChosen(final Size size, final int
rotation);

  protected abstract int getLayoutId();

  protected abstract Size getDesiredPreviewFrameSize();

  protected abstract void onInferenceConfigurationChanged();

  @Override
  public void onClick(View v) {
    if (v.getId() == R.id.plus) {
      String threads = threadsTextView.getText().toString().trim();
      int numThreads = Integer.parseInt(threads);
      if (numThreads >= 9) return;
      setNumThreads(++numThreads);
      threadsTextView.setText(String.valueOf(numThreads));
    } else if (v.getId() == R.id.minus) {
      String threads = threadsTextView.getText().toString().trim();
      int numThreads = Integer.parseInt(threads);
      if (numThreads == 1) {
        return;
      }
      setNumThreads(--numThreads);
      threadsTextView.setText(String.valueOf(numThreads));
    }
  }

  @Override
  public void onItemSelected(AdapterView<?> parent, View view, int pos, long
id) {
    if (parent == modelSpinner) {

setModel(Model.valueOf(parent.getItemAtPosition(pos).toString().toUpperCase
()));
    } else if (parent == deviceSpinner) {
      setDevice(Device.valueOf(parent.getItemAtPosition(pos).toString()));
    }
  }

  @Override
  public void onNothingSelected(AdapterView<?> parent) {
    // Do nothing.
  }}
```

Question Answering Model Code

```
pip install transformers
from transformers import BertForQuestionAnswering, AutoTokenizer
modelname = 'mobilebert-uncased'
model = BertForQuestionAnswering.from_pretrained(modelname)
tokenizer = AutoTokenizer.from_pretrained(modelname)

from transformers import pipeline
nlp = pipeline('question-answering', model=model, tokenizer=tokenizer)
context = 'Preheat the oven to 350 degrees F (175 degrees C). Lightly
grease a 5x9 inch loaf pan.It is preferred to cook it in oil than in
water.\nPress the brown sugar in the bottom of the prepared loaf pan and
spread the ketchup over the sugar.\nIn a mixing bowl, mix thoroughly all
remaining ingredients and shape into a loaf. Place on top of the
ketchup.\nBake in a preheated oven for 1 hour or until the juices are
clear.'

nlp({
    'question': ' Should I cook it in Oil or in Water?',
    'context': context
})
```

Question Answering App Android Studio

Build Gradle

```
apply plugin: 'com.android.application'

android {
   compileSdkVersion 29
   buildToolsVersion "29.0.0"
   defaultConfig {
       applicationId "org.tensorflow.lite.examples.bertapp"
       minSdkVersion 26
       targetSdkVersion 29
       versionCode 1
       versionName "1.0"
```

```
        testInstrumentationRunner "androidx.test.runner.AndroidJUnitRunner"
    }
    buildTypes {
        release {
            minifyEnabled false
            proguardFiles
getDefaultProguardFile('proguard-android-optimize.txt'),
'proguard-rules.pro'
        }
    }
    aaptOptions {
        noCompress "tflite"
    }
    compileOptions {
        sourceCompatibility JavaVersion.VERSION_1_8
        targetCompatibility JavaVersion.VERSION_1_8
    }
    // If you find lint problem like:
    // * What went wrong:
    // A problem was found with the configuration of task ':app:lint'.
    // > No value has been specified for property 'lintClassPath'.
    //
    // Probably you haven't set ANDROID_HOME. To fix:
    // export ANDROID_HOME=$HOME/Android/Sdk   # Your Android SDK path.
    lintOptions {
      abortOnError false
    }
}

// Download TF Lite model.
apply from: 'download.gradle'

dependencies {
    implementation fileTree(dir: 'libs', include: ['*.jar'])

    implementation 'com.android.support.constraint:constraint-layout:1.1.3'
    implementation 'com.android.support:appcompat-v7:28.0.0'
    implementation 'com.android.support:support-annotations:28.0.0'
    implementation 'com.android.support:support-v4:28.0.0'
    implementation 'com.android.support:recyclerview-v7:28.0.0'
    implementation 'com.android.support:design:28.0.0'

    implementation 'com.google.code.gson:gson:2.8.5'
    implementation 'com.google.guava:guava:28.1-android'
    implementation 'org.tensorflow:tensorflow-lite:2.2.0'

    testImplementation 'junit:junit:4.12'
    testImplementation 'androidx.test:core:1.2.0'
    testImplementation 'com.google.truth:truth:1.0'
    testImplementation 'org.robolectric:robolectric:4.3.1'
    androidTestImplementation 'androidx.test:runner:1.2.0'
    androidTestImplementation 'androidx.test.espresso:espresso-core:3.2.0'
}
```

QA Activity

```
package org.tensorflow.lite.examples.bertqa.ui;

import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.HandlerThread;
import android.speech.tts.TextToSpeech;
import android.support.v7.app.AppCompatActivity;
import android.support.v7.widget.LinearLayoutManager;
import android.support.v7.widget.RecyclerView;
import android.text.Editable;
import android.text.Spannable;
import android.text.SpannableString;
import android.text.TextWatcher;
import android.text.method.ScrollingMovementMethod;
import android.text.style.BackgroundColorSpan;
import android.util.Log;
import android.view.KeyEvent;
import android.view.View;
import android.view.inputmethod.InputMethodManager;
import android.widget.ImageButton;
import android.widget.TextView;
import android.support.design.widget.Snackbar;
import android.support.design.widget.TextInputEditText;
import java.util.List;
import java.util.Locale;
import org.tensorflow.lite.examples.bertqa.R;
import org.tensorflow.lite.examples.bertqa.ml.LoadDatasetClient;
import org.tensorflow.lite.examples.bertqa.ml.QaAnswer;
import org.tensorflow.lite.examples.bertqa.ml.QaClient;

/** Activity for doing Q&A on a specific dataset */
public class QaActivity extends AppCompatActivity {

 private static final String DATASET_POSITION_KEY = "DATASET_POSITION";
 private static final String TAG = "QaActivity";
 private static final boolean DISPLAY_RUNNING_TIME = false;

 private TextInputEditText questionEditText;
 private TextView contentTextView;
 private TextToSpeech textToSpeech;

 private boolean questionAnswered = false;
 private String content;
 private Handler handler;
 private QaClient qaClient;

 public static Intent newInstance(Context context, int datasetPosition) {
```

```
    Intent intent = new Intent(context, QaActivity.class);
    intent.putExtra(DATASET_POSITION_KEY, datasetPosition);
    return intent;
  }

  @Override
  protected void onCreate(Bundle savedInstanceState) {
    Log.v(TAG, "onCreate");
    super.onCreate(savedInstanceState);
    setContentView(R.layout.tfe_qa_activity_qa);

    // Get content of the selected dataset.
    int datasetPosition = getIntent().getIntExtra(DATASET_POSITION_KEY, -1);
    LoadDatasetClient datasetClient = new LoadDatasetClient(this);

    // Show the dataset title.
    TextView titleText = findViewById(R.id.title_text);
    titleText.setText(datasetClient.getTitles()[datasetPosition]);

    // Show the text content of the selected dataset.
    content = datasetClient.getContent(datasetPosition);
    contentTextView = findViewById(R.id.content_text);
    contentTextView.setText(content);
    contentTextView.setMovementMethod(new ScrollingMovementMethod());

    // Setup question suggestion list.
    RecyclerView questionSuggestionsView =
findViewById(R.id.suggestion_list);
    QuestionAdapter adapter =
        new QuestionAdapter(this,
datasetClient.getQuestions(datasetPosition));
    adapter.setOnQuestionSelectListener(question ->
answerQuestion(question));
    questionSuggestionsView.setAdapter(adapter);
    LinearLayoutManager layoutManager =
        new LinearLayoutManager(this, LinearLayoutManager.HORIZONTAL,
false);
    questionSuggestionsView.setLayoutManager(layoutManager);

    // Setup ask button.
    ImageButton askButton = findViewById(R.id.ask_button);
    askButton.setOnClickListener(
        view -> answerQuestion(questionEditText.getText().toString()));

    // Setup text edit where users can input their question.
    questionEditText = findViewById(R.id.question_edit_text);
    questionEditText.setOnFocusChangeListener(
        (view, hasFocus) -> {
          // If we already answer current question, clear the question so
that user can input a new
          // one.
          if (hasFocus && questionAnswered) {
            questionEditText.setText(null);
```

```
        }
      });
  questionEditText.addTextChangedListener(
      new TextWatcher() {
        @Override
        public void beforeTextChanged(CharSequence charSequence, int i,
int i1, int i2) {}

        @Override
        public void onTextChanged(CharSequence charSequence, int i, int
i1, int i2) {
          // Only allow clicking the Ask button if there is a question.
          boolean shouldAskButtonActive =
!charSequence.toString().isEmpty();
          askButton.setClickable(shouldAskButtonActive);
          askButton.setImageResource(
              shouldAskButtonActive ? R.drawable.ic_ask_active :
R.drawable.ic_ask_inactive);
        }

        @Override
        public void afterTextChanged(Editable editable) {}
      });
  questionEditText.setOnKeyListener(
      (v, keyCode, event) -> {
        if (event.getAction() == KeyEvent.ACTION_UP && keyCode ==
KeyEvent.KEYCODE_ENTER) {
          answerQuestion(questionEditText.getText().toString());
        }
        return false;
      });

  // Setup QA client to and background thread to run inference.
  HandlerThread handlerThread = new HandlerThread("QAClient");
  handlerThread.start();
  handler = new Handler(handlerThread.getLooper());
  qaClient = new QaClient(this);
}

@Override
protected void onStart() {
  Log.v(TAG, "onStart");
  super.onStart();
  handler.post(
      () -> {
        qaClient.loadModel();
        qaClient.loadDictionary();
      });

  textToSpeech =
      new TextToSpeech(
          this,
          status -> {
```

```
            if (status == TextToSpeech.SUCCESS) {
              textToSpeech.setLanguage(Locale.US);
            } else {
              textToSpeech = null;
            }
          });
  }

  @Override
  protected void onStop() {
    Log.v(TAG, "onStop");
    super.onStop();
    handler.post(() -> qaClient.unload());

    if (textToSpeech != null) {
      textToSpeech.stop();
      textToSpeech.shutdown();
    }
  }

  private void answerQuestion(String question) {
    question = question.trim();
    if (question.isEmpty()) {
      questionEditText.setText(question);
      return;
    }

    // Append question mark '?' if not ended with '?'.
    // This aligns with question format that trains the model.
    if (!question.endsWith("?")) {
      question += '?';
    }
    final String questionToAsk = question;
    questionEditText.setText(questionToAsk);

    // Delete all pending tasks.
    handler.removeCallbacksAndMessages(null);

    // Hide keyboard and dismiss focus on text edit.
    InputMethodManager imm =
        (InputMethodManager)
getSystemService(AppCompatActivity.INPUT_METHOD_SERVICE);
    imm.hideSoftInputFromWindow(getWindow().getDecorView().getWindowToken(),
0);
    View focusView = getCurrentFocus();
    if (focusView != null) {
      focusView.clearFocus();
    }

    // Reset content text view
    contentTextView.setText(content);

    questionAnswered = false;
```

```
   Snackbar runningSnackbar =
       Snackbar.make(contentTextView, "Looking up answer...",
Integer.MAX_VALUE);
   runningSnackbar.show();

   // Run TF Lite model to get the answer.
   handler.post(
       () -> {
         long beforeTime = System.currentTimeMillis();
         final List<QaAnswer> answers = qaClient.predict(questionToAsk,
content);
         long afterTime = System.currentTimeMillis();
         double totalSeconds = (afterTime - beforeTime) / 1000.0;

         if (!answers.isEmpty()) {
           // Get the top answer
           QaAnswer topAnswer = answers.get(0);
           // Show the answer.
           runOnUiThread(
               () -> {
                 runningSnackbar.dismiss();
                 presentAnswer(topAnswer);

                 String displayMessage = "Top answer was successfully
highlighted.";
                 if (DISPLAY_RUNNING_TIME) {
                   displayMessage = String.format("%s %.3fs.",
displayMessage, totalSeconds);
                 }
                 Snackbar.make(contentTextView, displayMessage,
Snackbar.LENGTH_LONG).show();
                 questionAnswered = true;
               });
         }
       });
 }

 private void presentAnswer(QaAnswer answer) {
   // Highlight answer.
   Spannable spanText = new SpannableString(content);
   int offset = content.indexOf(answer.text, 0);
   if (offset >= 0) {
     spanText.setSpan(
         new BackgroundColorSpan(getColor(R.color.tfe_qa_color_highlight)),
         offset,
         offset + answer.text.length(),
         Spannable.SPAN_EXCLUSIVE_EXCLUSIVE);
   }
   contentTextView.setText(spanText);

   // Use TTS to speak out the answer.
   if (textToSpeech != null) {
```

```
     textToSpeech.speak(answer.text, TextToSpeech.QUEUE_FLUSH, null,
answer.text);
   }
 }
}
```

Tensorflow Model convert to Lite model

```
from google.colab import drive
drive.mount('/content/drive',force_remount=True)
import os
import numpy as np
import tensorflow as tf
from tensorflow.keras.models import load_model
from tensorflow.keras.preprocessing.image import ImageDataGenerator
from tensorflow.keras.applications.mobilenet import preprocess_input
# configuration parameters
TEST_DATA_DIR = './test/'
MODEL_PATH = '/content/drive/MyDrive/bestmodel_10class.hdf5'
TFLITE_MODEL_DIR = "./models/tflite/"
TEST_SAMPLES = 450
IMG_WIDTH, IMG_HEIGHT = 224, 224
TFLITE_MODEL_DIR = "./models/tflite/"
# convert a tf.Keras model to tflite model with INT8 quantization
# Note: INT8 quantization is by default!
My_TFlite_Model =
tf.keras.models.load_model('/content/drive/MyDrive/bestmodel_10class.hdf5')
converter = tf.lite.TFLiteConverter.from_keras_model(My_TFlite_Model)
converter.optimizations = [tf.lite.Optimize.OPTIMIZE_FOR_SIZE]
tflite_model = converter.convert()

# write the model to a tflite file as binary file
```

```
tflite_no_quant_file = "model_quant.tflite"
with open("model_quant.tflite", "wb") as f:
    f.write(tflite_model)

# Note: you should see roughly 4x times reduction in the model size
My_TFlite_Model =
tf.keras.models.load_model('/content/drive/MyDrive/bestmodel_10class.hdf5')
converter = tf.lite.TFLiteConverter.from_keras_model(My_TFlite_Model)
tflite_model = converter.convert()
open("My_TFlite_Model.tflite", "wb").write(tflite_model)
```

Chapter 7

REFERENCES

[1] A. B. Ocay, J. M. Fernandez and T. D. Palaoag, "NutriTrack: Android-based food recognition app for nutrition awareness," 2017 3rd IEEE International Conference on Computer and Communications (ICCC), 2017, pp. 2099-2104, doi: 10.1109/CompComm.2017.8322907.

[2] Christodoulidis, Stergios & Anthimopoulos, Marios & Mougiakakou, Stavroula. (2015). Food Recognition for Dietary Assessment Using Deep Convolutional Neural Networks. 9281. 458-465. 10.1007/978-3-319-23222-5_56.

[3] Manna R., Das D., Gelbukh A. (2020) Information Retrieval-Based Question Answering System on Foods and Recipes. In: Martínez-Villaseñor L., Herrera-Alcántara O., Ponce H., Castro-Espinoza F.A. (eds) Advances in Computational Intelligence. MICAI 2020. Lecture Notes in Computer Science, vol 12469. Springer, Cham. https://doi.org/10.1007/978-3-030-60887-3_23

[4] Impact of covid on obesity https://www.npr.org/sections/health-shots/2021/09/29/1041515129/obesity-rates-rise-during-pandemic-fueled-by-stress-job-loss-sedentary-lifestyle

[5] statistics of obesity and diabetes https://www.a-mansia.com/obesity-and-diabetes-in-the-world/

[6] FOOD-101 dataset https://www.kaggle.com/dansbecker/food-101

[7] FOOD-20 dataset https://www.kaggle.com/cdart99/food20dataset

[8] Tensorflow Inception V3 documentation https://www.tensorflow.org/api_docs/python/tf/keras/applications/inception_v3/InceptionV3

[9] BERT documentation https://huggingface.co/docs/transformers/index

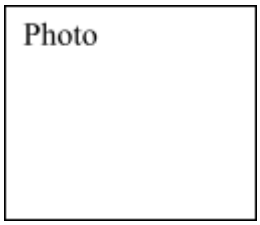
[10] MobileBERT documentation https://huggingface.co/docs/transformers/model_doc/mobilebert

[11] Tensorflow Lite Guide https://www.tensorflow.org/lite/guide

[12] Skeleton UI for Food Detection App and Q&A App https://github.com/tensorflow/examples

[13] Recipe-Box dataset https://eightportions.com/datasets/Recipes/

BIODATA



Name : L.Satyajit
Mobile Number : 8637013050
E-mail : satyajit.18bcd7143@vitap.ac.in
Permanent Address : T-6,3-C, Salarpuria Gardenia, Bidhan Nagar, Durgapur, West Bengal, India. Pin- 713212.