CALORIE DETECTOR SYSTEM

A PROJECT REPORT

Submitted by

M.UDAYASHANKAR (913119205049)

In partial fulfillment for the award of the degree

of

BACHELORS OF TECHNOLOGY

IN

INFORMATION TECHNOLOGY

VELAMMAL COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRAGANOOR, MADURAI

ANNA UNIVERSITY: CHENNAI 600 025

SEPTEMPER 2020

VELAMMAL COLLEGE OF ENGINEERING AND TECHNOLOGY, VIRAGANOOR, MADURAI

DEPARTMENT OF INFORMATION TECHNOLOGY

ANNA UNIVERSITY: CHENNAI 600 025

BONAFIDE CERTIFICATE

Certified that this project report "**CALORIE DETECTOR SYSTEM"** is the bonafide work of "**M.UDAYA SHANKAR (913119205049),"** who carried out the project work under my supervision.

SIGNATURE	**SIGNATURE**
Dr.R.Perumalraja	**Mrs.M.Prabha**
HEAD OF THE DEPARTMENT	**SUPERVISOR**
Professor and Head	Assistant professor III
Information Technology Velammal College of Engineering and Technology	Information Technology Velammal College of Engineering and Technology

Submitted to the Project work and viva-voce Examination held on…….

Internal Examiner **External Examiner**

ACKNOWLEDGEMENT

I express my deep sense of gratitude to **Dr.N. Suresh Kumar**, Principal, Velammal College of Engineering and Technology for permitting and providing the facilities in carrying this project.

I would especially like to express my extreme gratitude and sincere thanks to **Dr.R.Perumalraja**, Professor and Head of the Department, Information Technology, Velammal College of Engineering and Technology for his enthusiastic and innovative guidance during the entire period of my project work.

I would like to express my deep sense of gratitude and heartiest thanks to my project guide **Mrs.M.Prabha**, Assisstant Professor III, Department of Information Technology Velammal College of Engineering and Technology for her continuous encouragement and valuable guidance. Her support made me to complete my work successfully.

I express my thanks to all teaching and non-teaching staff members of the Department of Information Technology for their cooperation during my project work.

Finally I would like to express my loving gratitude to my parents for the everlasting love and encouragement given by them during this period.

TABLE OF CONTENT

ABSTRACT

The process of identifying food items from an image is quite interesting. Food monitoring plays a leading role in health-related problems, and so it is becoming more essential nowadays.

Our proposed system allows the user to take a picture of the food and display the calorie intake, through their smartphones which is more convenient for users. In this paper, convolutional neural networks have been used to classify images of food.

Unlike the traditional artificial neural networks, convolutional neural networks have the capability of estimating the score function directly from image pixels. Bitmapping is done to get the image in fixed dimensions. Tensor Flow is used to mine the data set.

LIST OF FIGURES

CHAPTER 1: INTRODUCTION:

IMAGE PROCESSING:

Image processing is a method to convert an image into digital form and perform some operations on it, in order to get an enhanced image or to extract some useful information from it. It is a type of signal dispensation in which input is image, like video frame or photograph and output may be image or characteristics associated with that image. Usually Image Processing system includes treating images as two-dimensional signals while applying already set signal processing methods to them. It is among rapidly growing technologies today, with its applications in various aspects of a business. Image Processing forms core research area within engineering and computer science disciplines too.

Image processing basically includes the following three steps: Importing the image to the system. Analyzing and manipulating the image which includes data compression and image enhancement and spotting patterns.Output is the last stage in which result can be altered image or report that is based on image analysis.

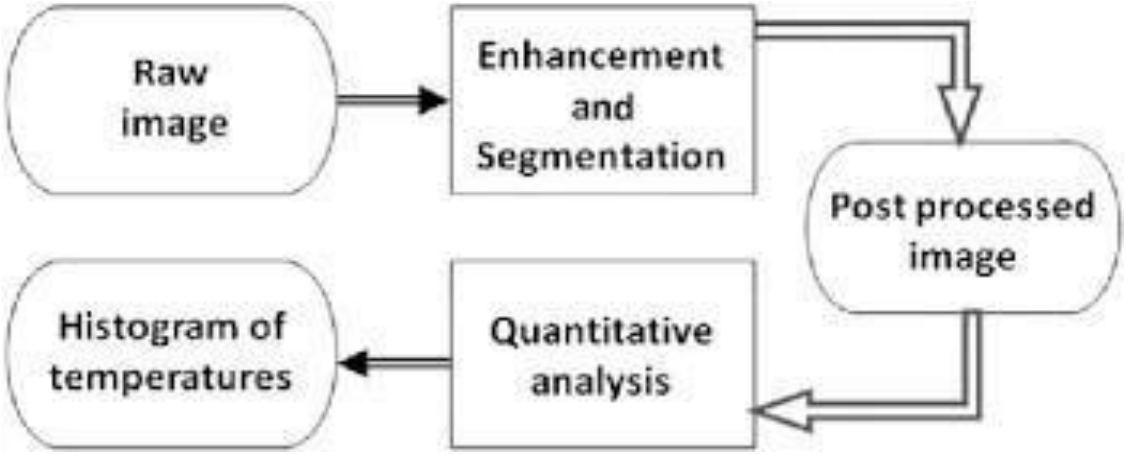


FIG 1.1.1.1 IMAGE PROCESSING

- **ARTIFICIAL INTELLIGENCE**

"Artificial Intelligence (AI) is the part of computer science concerned with designing intelligent computer systems, that is, systems that exhibit characteristics we associate with intelligence in human behaviour – understanding language, learning, reasoning, solving problems, and so on."

Scientific Goal To determine which ideas about knowledge representation,learning, rule systems, search, and so on, explain various sorts of real intelligence.Engineering Goal To solve real world problems using AI techniques such asknowledge representation, learning, rule systems, search, and so on. Traditionally, computer scientists and engineers have been more interested in the engineering goal, while psychologists, philosophers and cognitive

scientists have been more interested in the scientific goal.

The Roots - Artificial Intelligence has identifiable roots in a number of older disciplines, particularly – Philosophy, Logic/Mathematics, Computation, Logic/Mathematics, Computation Psychology/Cognitive Science, Biology/Neuroscience, Evolution.

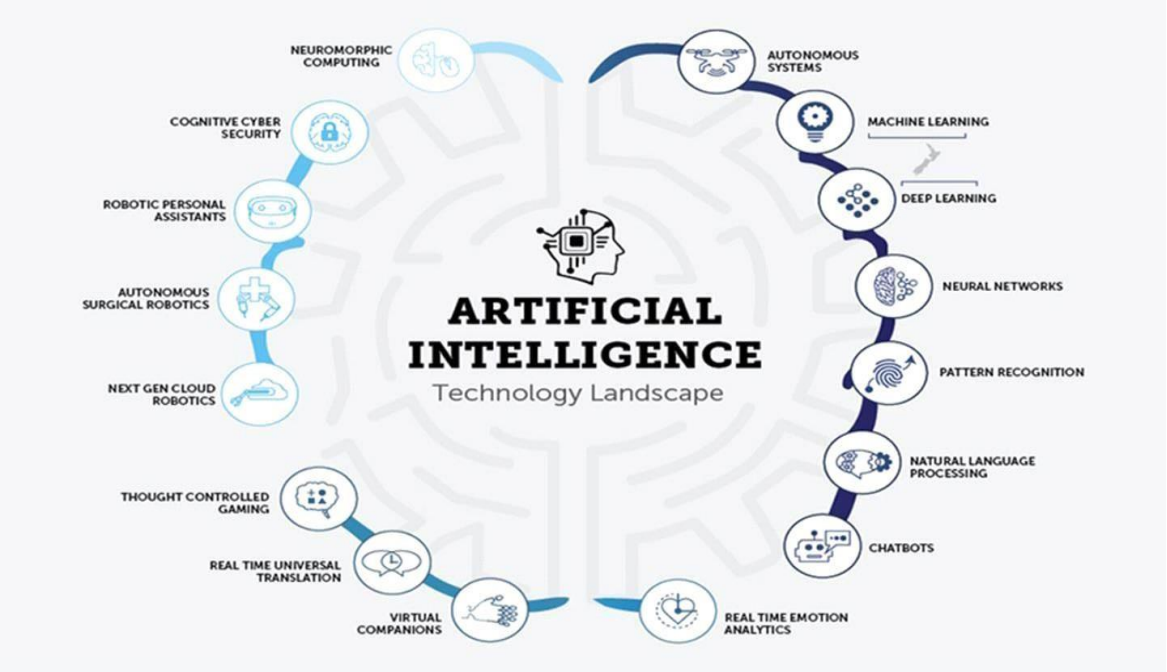


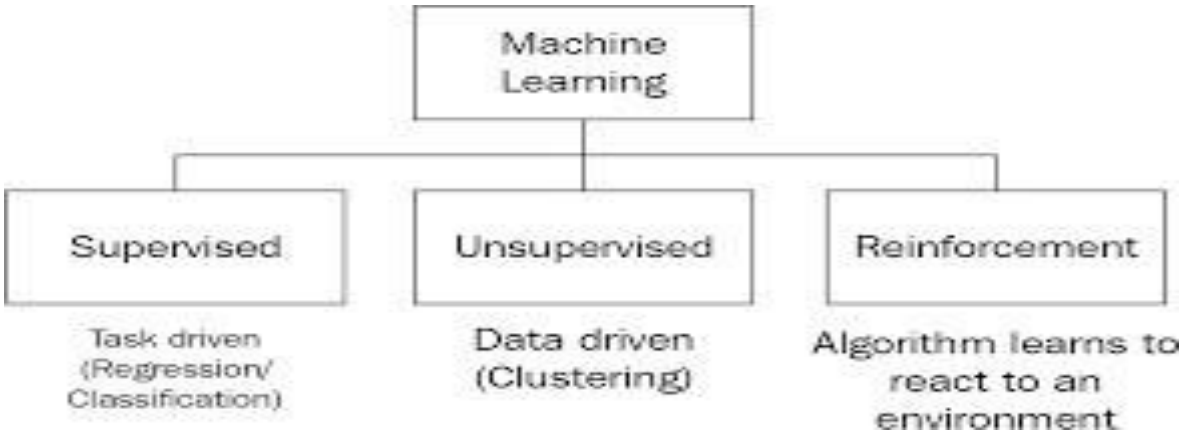
FIG 1.1.2.1 ARTIFICIAL INTELLIGENCE

- **MACHINE LEARNING**

Machine learning is a subset of artificial intelligence in the field of computer science that often uses statistical techniques to give computers the ability to "learn" (i.e., progressively improve performance on a specific task) with data, without being explicitly programmed.is a type of Artificial Intelligence that provides computers with the ability to learn without being explicitly programmed.

Machine Learning Approaches

Supervised Learning: Learning with a labelled training set.
Example: Email spam detector with training set of labelled emails.
Unsupervised Learning: Discovering patterns in un-labelled data.
Example: Cluster similar documents based on the text content.
Reinforcement Learning: Learning based on feedback or reward.
Example: Learn to play chess by winning or losing.



Neural Networks

When we say "Neural Networks", we mean artificial Neural Networks (ANN). The idea of ANN is based on biological neural networks like the brain.The basic structure of a neural network is the neuron. A neuron in biology consists of three major parts: the soma (cell body), the dendrites, and the axon.The dendrites branch of from the soma in a tree-like way and getting thinner with every branch. They receive signals (impulses) from other neurons at synapses. The axon - there is always only one - also leaves the soma and usually tend to extend for longer distances than the dendrites. The axon is used for sending the output of the neuron to other neurons or better to the synapsis of other neurons. When a signal comes in, it gets multiplied by a weight value that is assigned to this particular input. That is, if a neuron has three inputs, then it has three weights that can be adjusted individually. The weights usually get adjusted during the learn phase. After this the modified input signal are summed up. It is also possible to add additionally a so-called bias b to this sum. The bias is a value which can also be adjusted during the learn phase.Finally, the actual output has to be determined. For this purpose, an activation or step function Φ is applied to weighted sum of the input values.

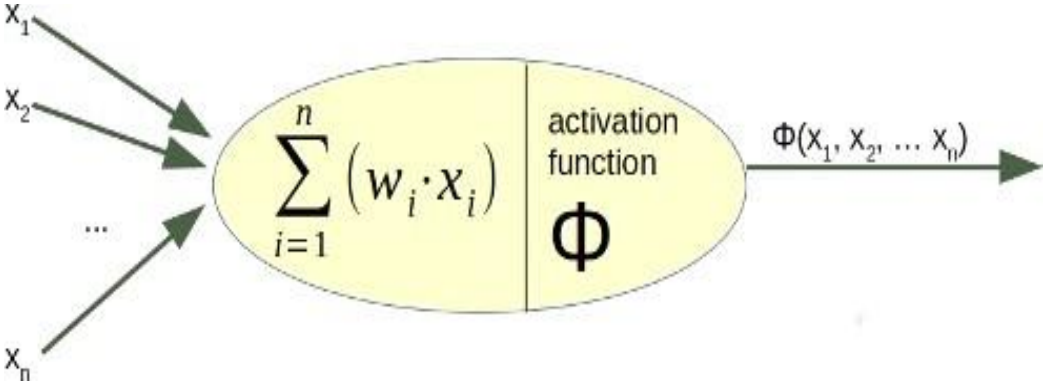


FIG 1.1.3.2 NEURAL NETWORK

A hidden layer allows the network to reorganize or rearrange the input data.

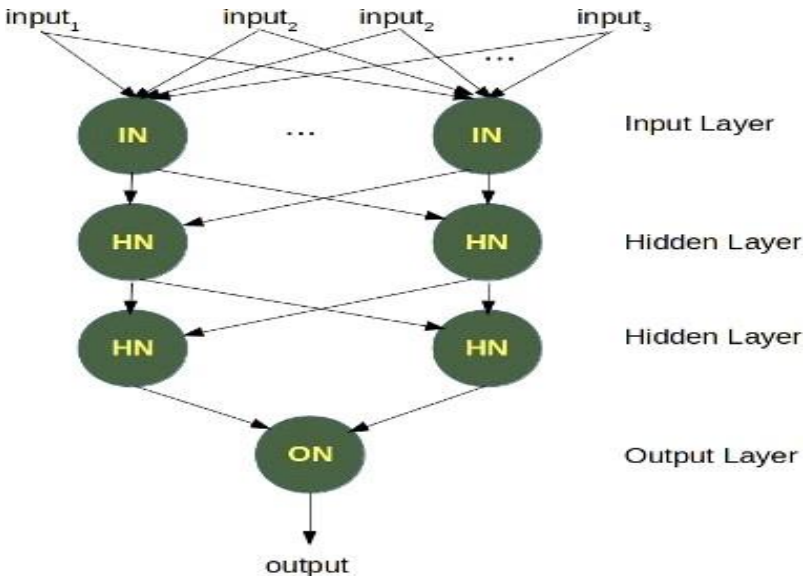


FIG 1.1.3.3 NEURAL NETWORK STRUCTURE

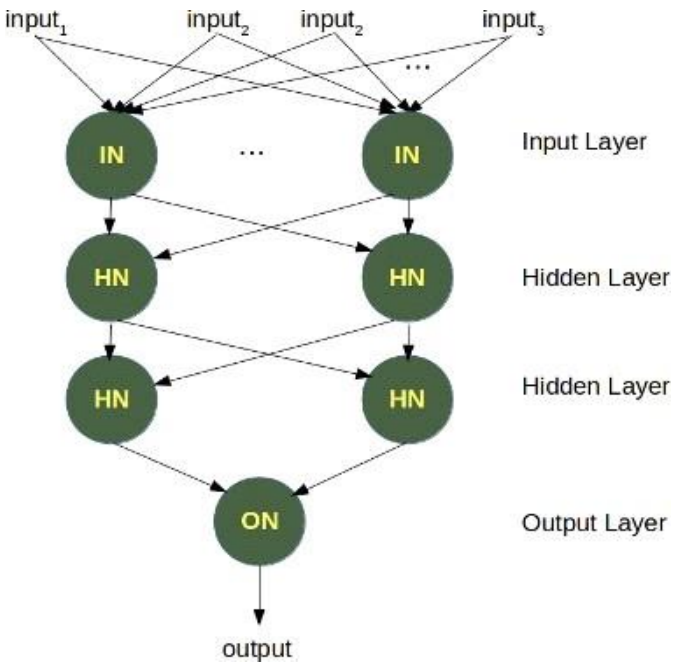


FIG 1.1.3.4 NEURAL NETWORK ARCHITECTURE

DEEP NEURAL NETWORK:

A deep neural network is a neural network with a certain level of complexity, a neural network with more than two layers. Deep neural networks use sophisticated mathematical modeling to process data in complex ways. A neural network, in general, is a technology built

to simulate the activity of the human

brain – specifically, pattern recognition and the passage of input through various layers of simulated neural connections.

Many experts define deep neural networks as networks that have an input layer, an output layer and at least one hidden layer in between. Each layer performs specific types of sorting and ordering in a process that some refer to as "feature hierarchy." One of the key uses of these sophisticated neural networks is dealing with unlabelled or unstructured data. The phrase "deep learning" is also used to describe these deep neural networks, as deep learning represents a specific form of machine learning where technologies using aspects of artificial intelligence seek to classify and order information in ways that go beyond simple input/output protocols.

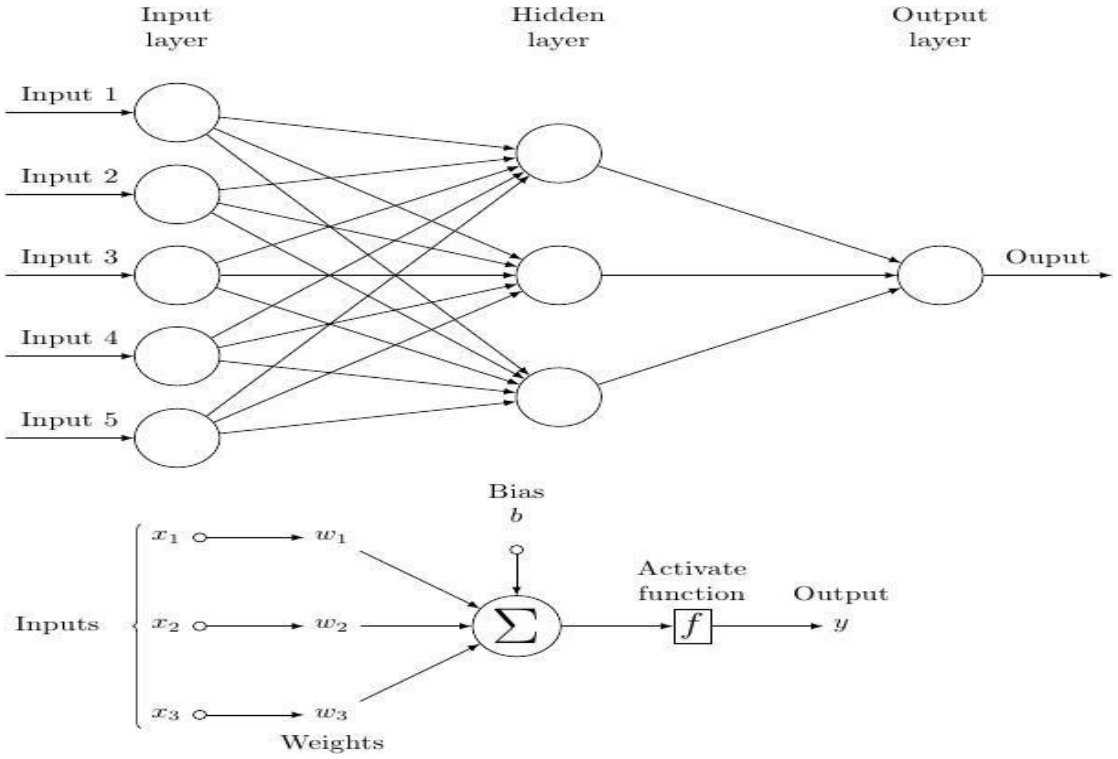


FIG 1.1.3.5 DEEP NEURAL NETWORK

```
# Adding the input layer and the first hidden layer
classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu',
input_dim = 11))
# Adding the second hidden layer

classifier.add(Dense(output_dim = 6, init = 'uniform', activation = 'relu')) #
```

Adding the output layer
classifier.add(Dense(output_dim = 1, init = 'uniform', activation = 'sigmoid'))

First argument is Optimizer, this is the algorithm we use to find optimal set of weights. This algorithm is Stochastic Gradient descent (SGD). Among several types of SGD algorithm, the one which we will use is 'Adam'. If we go in deeper detail of SGD, we will find that SGD depends on loss thus our second parameter is loss. Since out dependent variable is binary, we will have to use logarithmic loss function called 'binary cross entropy', if our dependent variable has more than 2 categories in output then use 'categorical_crossentropy'. We want to improve performance of our neural network based on accuracy so add metrics as accuracy.

We can train our model on training data. We use fit method to the fit our model Batch size is used to specify the number of observations after which we want to update weight. Epoch is nothing but the total number of iterations. Choosing the value of batch size and epoch is trial and error there is no specific rule for that.

- **CONVOLUTIONAL NEURAL NETWORK**

CNNs use a variation of multilayer perceptrons designed to require minimal pre-processing. They are also known as Shift Invariant or Space Invariant Artificial Neural networks (SIANN), based on their shared-weights architecture and translation invariance characteristics.

Convolutional networks were inspired by biological processes in that the connectivity pattern between neurons resembles the organization of the animal visual cortex. Individual cortical neurons respond to stimuli only in a restricted region of the visual field known as the receptive field. The receptive fields of different neurons partially overlap such that they cover the entire visual field.

CNNs use relatively little pre-processing compared to other image classification algorithms. This means that the network learns the filters that in traditional algorithms were hand-engineered. This independence from prior knowledge and human effort in feature design is a major advantage.

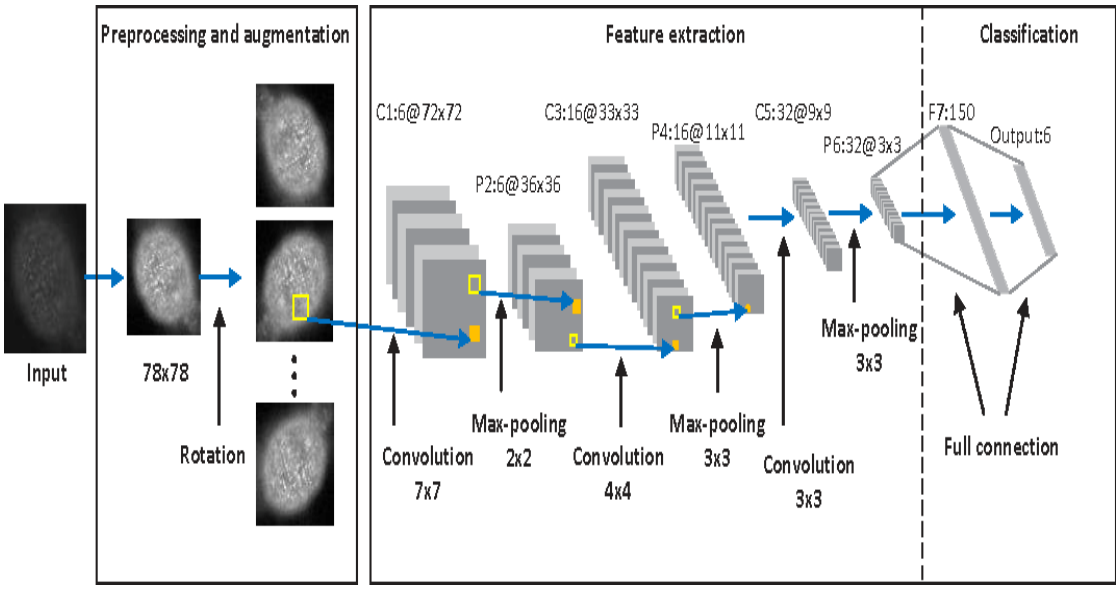
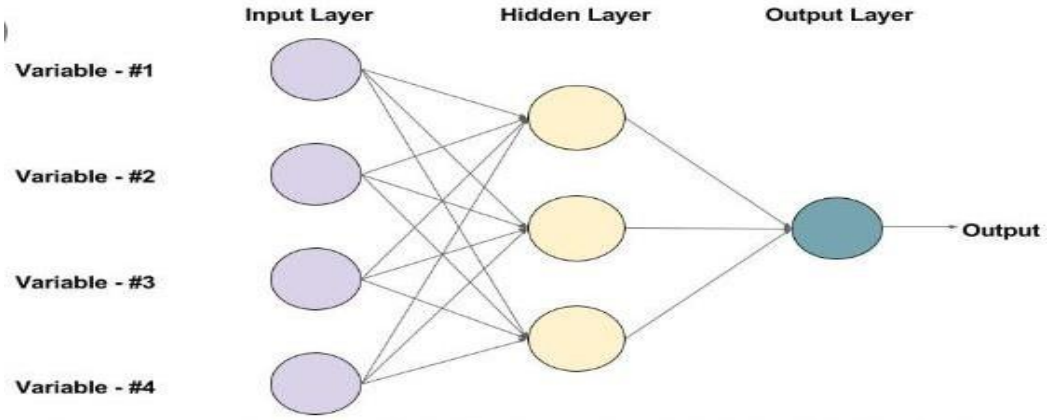


FIG 1.1.4.1 CONVOLUTIONAL NEURAL NETWORK

Feedforward neural networks:

Feed forward neural networks are artificial neural networks where the connections between units do not form a cycle. Feedforward neural networks were the first type of artificial neural network invented and are simpler than their counterpart, recurrent neural networks. They are called feedforward because information only travels forward in the network (no loops), first through the input nodes, then through the hidden nodes (if present), and finally through the output nodes.

Feedfoward neural networks are primarily used for supervised learning in cases where the data to be learned is neither sequential nor time-dependent. That is, feedforward neural networks compute a function fff on fixed size input xxx such that f(x)≈yf(x) \approx yf(x)≈y for training pairs (x,y)(x, y)(x,y). On the other hand, recurrent neural networks learn sequential data, computing ggg on variable length input Xk={x1,…,xk}X_k = \{x_1, \dots, x_k\}Xk={x1,…,xk} such that g(Xk)≈ykg(X_k) \approx y_kg(Xk)≈yk for training pairs (Xn,Yn)(X_n, Y_n)(Xn,Yn) for all 1≤k≤n1 \le k \le n1≤k≤n.



An example of a Feed-forward Neural Network with one hidden layer (with 3 neurons)

CHAPTER 2: LITERATURE SURVEY:

Food Calorie Measurement Using Deep Learning Neural Network:

Accurate methods to measure food and energy intake are crucial for the battle against obesity. Providing users/patients with convenient and intelligent solutions that help them measure their food intake and collect dietary information are the most valuable insights toward long-term prevention and successful treatment programs. In this paper, we propose an assistive calorie measurement system to help patients and doctors succeed in their fight against diet-related health conditions. Our proposed system runs on smartphones, which allow the user to take a picture of the food and measure the amount of calorie intake automatically. In order to identify the food accurately in the system, we use deep convolutional neural networks to classify 10000 high-resolution food images for system training. Our results show that the accuracy of our method for food recognition of single food portions is 99%. The analysis and implementation of the proposed system are also described in this paper.

Smart Management of Food Classi fication Using Deep Learning:

Deep learning has been proved to be an advanced technology for big data analysis with a large number of successful cases in image processing, speech recognition, object detection, and so on.Recently ,it has also been introduced in food science and engineering.To our knowledge, this review is the first in the food domain.In this paper, we provided a brief introduction of deep learning and detailedly described the structure of some popular architectures of deep neural networks and the approaches for training a model.We surveyed dozens of articles that used deeplearning as the data analysis tool to solve the problems and challenges in food domain, including food recognition, calories estimation, quality detection of fruits, vegetables ,meat and aquatic products, food supply chain, and food contamination.The specific problems, the datasets, the preprocessing methods, the networks and frameworks used, the performance achieved, and the comparison with other popular solutions of each research were investigated. We also analyzed the

potential of deep learning to be used as an advanced data mining tool in food sensory and consume researches. The result of our survey indicates that deep learning out performs other methods such as manual feature extractors, conventional machine learning algorithms, and deep learning as a promising tool in food quality and safety inspection.The encouraging results in classification and regression problems achievd by deep learning will at act more research efforts to apply deep learning in to the field of food in the future.

Application of Deep Learning in Food:

Deep learning has been proved to be an advanced technology for big data analysis with a large number of successfulcases inimage processing, speechrecognition,objectdetection,and so on.Recently,ithas also been introduced in food science and engineering. To our knowledge, this review is the first in the food domain. In this paper, we provided a brief introduction of deep learning and detailedly described the structure of some popular architecture of deep neural networks and the approaches for training a model. We surveyed dozens of articles that used deep learning as the data analysis tool to solve the problems and challenges in food domain, including food recognition, calories estimation, quality detection of fruits, vegetables, meat and aquatic products, food supply chain, and food contamination. The specific problems, the datasets, the preprocessing methods, the networks and frameworks used, the performance achieved, and the comparison with other popular solutions of each research were investigated. We also analyzed the potential of deep learning to be used as an advanced data mining tool in food sensory and consume researches. The result of our survey indicates that deep learning outperforms other methods such as manual feature extractors, conventional machine learning algorithms, and deep learning as a promising tool in food quality and safety inspection. The encouraging results in classification and regression problems achieved by deep learning will attract more research efforts to apply deep learning into the field of food in the future.

Food Classification from Images Using Convolutional Neural Networks:

The process of identifying food items from an image is quite an interesting field with various applications. Since food monitoring plays a leading role in health-related problems, it is becoming more essential in our day-to-day lives. In this paper, an approach has been presented to classify images of food using convolutional neural networks. Unlike the traditional artificialneuralnetworks, convolutional neural networkshavethe capability of estimating the score functions directly from image pixels. A 2D convolution layer has been utilised which creates a convolution kernel that is convolved with the layer input to produce a tensor of outputs. There are multiple such layers, and the outputs are concatenated at parts to form the final tensor of outputs. We also use the Max-Pooling function for the data, and the features extracted from this function are used to train the network. An accuracy of 86.97% for the classes of the FOOD-101 dataset is recognised using the proposed implementation. Index Terms—Convolution filters; Convolution layer; Convolutional neural networks; Food-101 dataset; Food classification; Image recognition; MAX pooling.

CHAPTER 3: SYSTEM ANALYSIS:

EXISTING SYSTEM:

There are some methods existing for dietary assessments which involve self-reporting and manually recorded instruments. But it has some issues with the evaluation of calorie consumption. calorie consumption by a participant is prone to be biased , i.e. underestimating and under reporting of food intake to increase the accuracy and to reduce the bias current methods are enhanced by mobile cloud computing system, which makes use of devices such as smartphones to capture dietary and calorie information. Next to this step, dietary and calorie information are analyzed by employing the computing capacity of the cloud for an assessment. However, users still have to enter the information manually. Though plenty of research and development efforts have been made in the field of visual-based dietary and calorie information analysis, the efficient extraction of information from food images remains a challenging issue.

Discreet cosine transform is used to portion the food and it is searched by segmentation process. To classify K-NN (k nearest neighbor) is used which takes much time to train the images and classify. If data is not assumed properly, data loss may occur.

PROPOSED SYSTEM:

In this paper, an effort has been made to classify the images of food for further diet monitoring applications using convolutional neural networks (CNNs). Since the CNNs are capable of handling a large amount of data and can estimate the features automatically, they have been utilized for the task of food classification. Convolutional Neural Networks along with a Global Average Pooling layer, generates Food Activation Maps (heat maps of food probability). Fine tuning is done for FAM generation, which includes adding a convolutional layer with stride, and setting a soft max layer. Additionally, via thresholding, bounding boxes are generated. The present work aims to combine some of the above methodologies together, that creates a food classification system that predicts the class of food the image is in, and also gives the calorie count based on the portion size visible. This concept has a high scope in the health sector, as people want to keep track of what and how much they eat and simplifying the process into the form of this implementation increases usage and awareness of health-related factors.

The steps in image processing are

1. Pre-processing and

2. Neural Network Training.

From this, the trained model can be obtained which will classify any supplied image based on the trained database.

Advantage:

1. Calorie detector Application is easy to use and understand. It isuser friendly.
2. There is no need of personal dietitian since user can check the calorie easily using this application.
3. There is no need to create an account, user can simply capture the food image and find the calorie using this application.
4. This application also has a calendar with which the user can view the previously captured food and their corresponding calories anytime.
5. As Tensor Flow is used in this application, the response time is high.
6. The captured images by the user are automatically saved in the user device.
7. This application also shows the accuracy of how much it has matched with the trained data set.
8. It not only predicts the calorie but also carbs, fat and protein of the captured food.
9. This application does not require internet connection.

Disadvantage:

1. It only gives about 90% accuracy in predicting the captured food.
2. Since there is no user id and password protection any one can view the details inside that application.
3. The System provides only the amount of calorie, fat, carbs, protein in the food captured. Hence the user may not know the limitation range for the intake of the above.

System Requirement:

- **SOFTWARE REQUIREMENTS**

A software requirement specification is a description of a softwaresystem to be developed. It lays out functional and non-functional requirements and it also describes the operating system and tool used in the system and they are:

Operating System	:	Windows
Coding Language	:	Python
Front End	:	Keras
Back End	:	TensorFlow & OpenCV package.

- **HARWARE REQUIREMENTS:**

Hardware specifications are technical description of the computer's components and capabilities. Processor speed, model and manufacturer, etc. So the hardware components

required for the proposed system are:

Processor	:	Intel i3
Hard Disk	:	1TB
RAM	:	4GB
Display Monitor	:	15.5”

- **LANGUAGE SPECIFICATION:**

- **PYTHON**

Python is a general-purpose interpreted, interactive, object-oriented, and high-level programming language. It was created by Guido van Rossum during 1985- 1990. Like Perl, Python source code is also available under the GNU

General Public License (GPL). Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages. Python is a high-level, interpreted, interactive and object-oriented scripting language. Python is designed to be highly readable. It uses English keywords frequently where as other languages use punctuation, and it has fewer syntactical constructions than other languages.

Python is Interpreted − Python is processed at runtime by the interpreter. You do not need to compile your program before executing it. This is similar to PERL and PHP.

Python is Interactive − You can actually sit at a Python prompt and interact with the interpreter directly to write your programs.

Python is Object-Oriented − Python supports Object-Oriented style or technique of programming that encapsulates code within objects.

Python is a Beginner's Language − Python is a great language for the beginner- level programmers and supports the development of a wide range of applications from simple text processing to WWW browsers to games.

History of Python

Python was developed by Guido van Rossum in the late eighties and early nineties at the National Research Institute for Mathematics and Computer Science in the Netherlands.Python is derived from many other languages, including ABC, Modula-3, C, C++, Algol-68, SmallTalk, and Unix shell and other scripting languages.Python is copyrighted. Like Perl, Python source code is now available under the GNU General Public License (GPL). Python is now maintained by a core development team at the institute, although Guido van Rossum still holds a vital role in directing its progress.

Features of Python

Python's features include −

Easy-to-learn − Python has few keywords, simple structure, and a clearly defined syntax. This allows the student to pick up the language quickly.

Easy-to-read − Python code is more clearly defined and visible to the eyes.

Easy-to-maintain − Python's source code is fairly easy-to-maintain.

A broad standard library − Python's bulk of the library is very portable and cross-platform compatible on UNIX, Windows, and Macintosh.

Interactive Mode − Python has support for an interactive mode which allows interactive testing and debugging of snippets of code.

Portable − Python can run on a wide variety of hardware platforms and has the same interface on all platforms.

Extendable − You can add low-level modules to the Python interpreter. These modules enable programmers to add to or customize their tools to be more efficient.

Databases − Python provides interfaces to all major commercial databases.

GUI Programming − Python supports GUI applications that can be created and ported to many system calls, libraries and windows systems, such as Windows MFC, Macintosh, and the X Window system of Unix.

Scalable − Python provides a better structure and support for large programs than shell scripting.

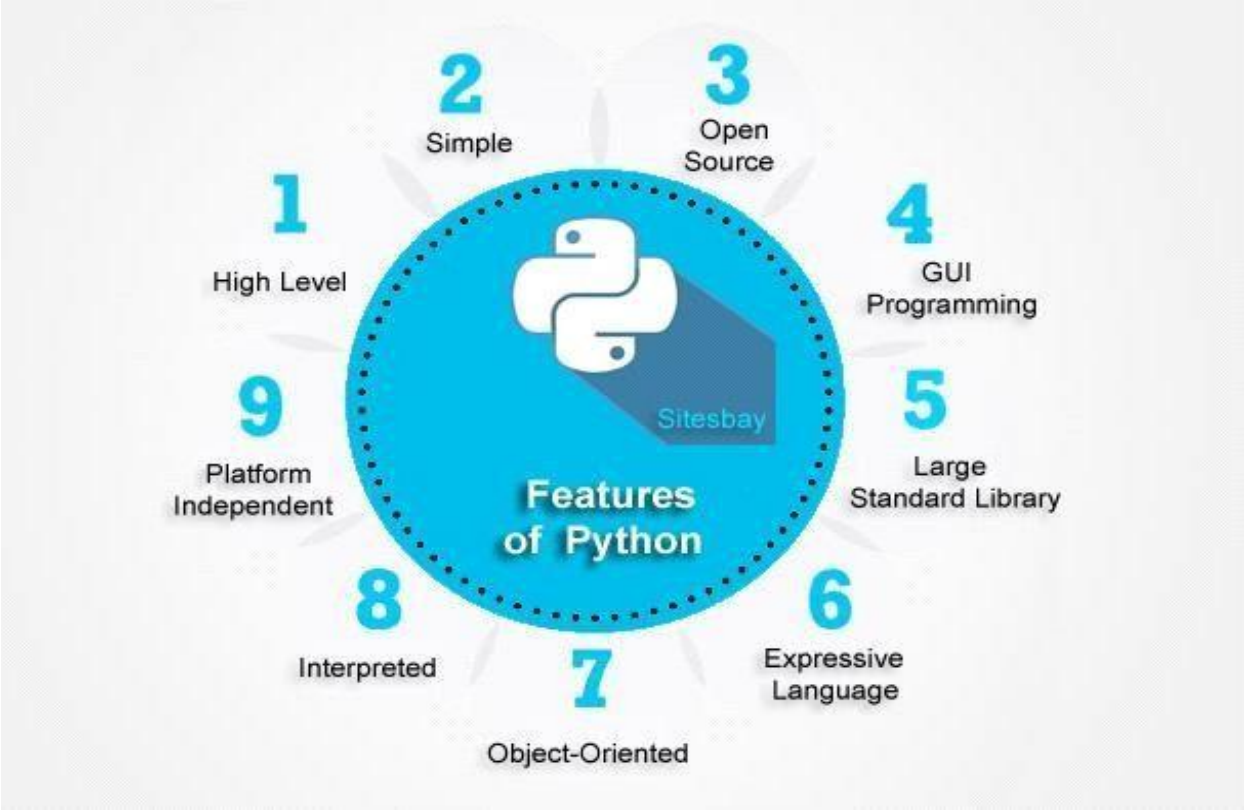


FIG 3.4.1.1 FEATURES OF PYTHON

Apart from the above-mentioned features, Python has a big list of good features, few are listed –

It supports functional and structured programming methods as well as OOP.It can be used as a scripting language or can be compiled to byte-code for building large applications.It provides very high-level dynamic data types and

supports dynamic type checking.It supports automatic garbage collection. It can be easily integrated with C, C++, COM, ActiveX, CORBA, and Java.

Python in Machine Learning:

Python is a popular platform used for research and development of production systems. It is a vast language with number of modules, packages and libraries that provides multiple ways of achieving a task. Python and its libraries like NumPy, SciPy, Scikit-Learn, Matplotlib are used in data science and data analysis. They are also extensively used for creating scalable machine learning algorithms. Python implements popular machine learning techniques such as Classification, Regression, Recommendation, and Clustering. Python offers ready- made framework for performing data mining tasks on large volumes of data effectively in lesser time. It includes several implementations achieved through algorithms such as linear regression, logistic regression, Naïve Bayes, k-means, K nearest neighbor, and

Random Forest.

Python has libraries that enable developers to use optimized algorithms. It implements popular machine learning techniques such as recommendation, classification, and clustering.

numpy − is used for its N-dimensional array objects.

pandas − is a data analysis library that includes data-frames.

matplotlib − is 2D plotting library for creating graphs and plots.

scikit-learn − the algorithms used for data analysis and data mining tasks.

seaborn − a data visualization library based on matplotlib.

Android Studio:

Android Studio is the official integrated development environment (IDE) for Google's Android operating system, built on JetBrains' IntelliJ IDEA software and designed specifically for Android development. It is available for download on Windows, macOS and Linux based operating systems. It is a replacement for the Eclipse Android Development Tools (ADT) as the primary IDE for native Android application development.

Android Studio was announced on May 16, 2013 at the Google I/O conference. It was in early access preview stage starting from version 0.1 in May 2013, then entered beta stage starting from version 0.8 which was released in June 2014. The first stable build was released in December 2014, starting from version 1.0.

On May 7, 2019, Kotlin replaced Java as Google's preferred language for Android app development. Java is still supported, as is C++.

Features:

The following features are provided in the current stable version:

- Gradle-based build support
- Android-specific refactoring and quick fixes
- Lint tools to catch performance, usability, version compatibility and other problems
- ProGuard integration and app-signing capabilities
- Template-based wizards to create common Android designs and components
- A rich layout editor that allows users to drag-and-drop UI components, option to preview layouts on multiple screen configurations.
- Support for building Android Wear apps
- Built-in support for Google Cloud Platform, enabling integration with Firebase Cloud

Messaging (Earlier 'Google Cloud Messaging') and Google App Engine.

- Android Virtual Device (Emulator) to run and debug apps in the Android studio.

Android Studio supports all the same programming languages of IntelliJ (and CLion) e.g. Java, C++, and more with extensions, such as Go;[19] and Android Studio 3.0 or later supports Kotlin and "all Java 7 language features and a subset of Java 8 language features that vary by platform version." External projects backport some Java 9 features. While IntelliJ that Android Studio is built on supports all released Java versions, and Java 12, it's not clear to what level Android Studio supports Java versions up to Java 12 (the documentation mentions partial Java 8 support). At least some new language features up to Java 12 areusable in Android.

VII. TENSOR FLOW OBJECT DETECTION API

The Tensor Flow Object Detection API is an open source framework built on top of Tensor Flow that makes it easy to construct, train and deploy object detection models. The Tensor Flow Object Detection API makes it extremely easy to train your own object detection model for a large variety of applications. With an object detection model, not only can you classify multiple objects in one image, but you can specify exactly where that object is in an image with a bounding box framing the object.

Step 1: Work with dataset:

The initial step is to generate a pre-trained model file with the help of CNN network. It is performed by initially capturing a set of images of one particular class (For e.g. 25 images of carrot class).

Step 2: Label the dataset:

Next step is labeling them with object name-set (object being carrot). These images are considered the set of relevant images. After the image-set are captured, the system is trained with these images. In our case, we trained the system with background images, so it does not recognize them or categorize them as part of the image class.

Step 3: Convert labels to binary format (TF Record):

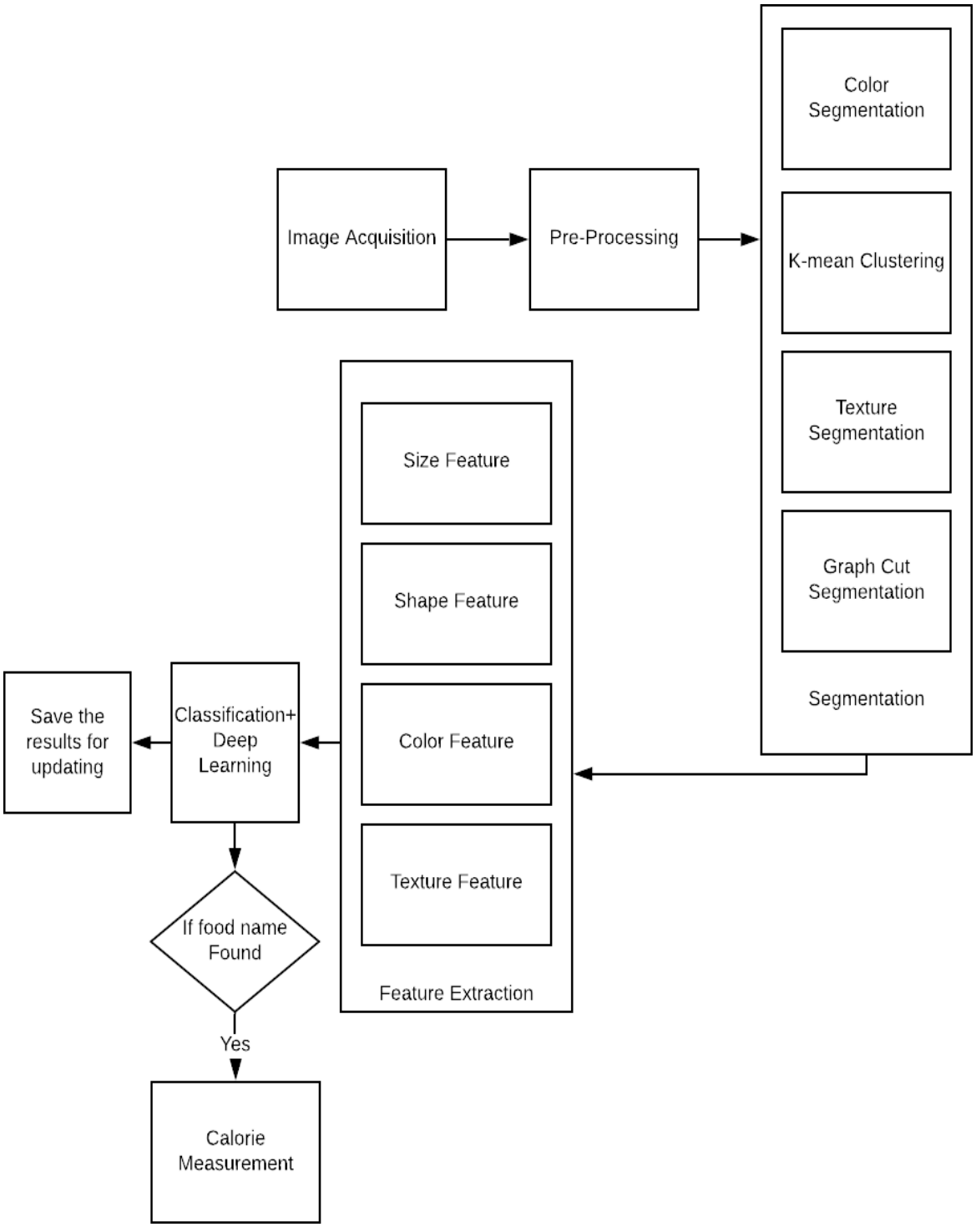
Using a binary file format for storage of our data can have a great impact on the performance of our import pipeline and as a consequence on the training time of our model. Binary data takes up less space on disk, takes less time to copy and can be read much more efficiently from disk. Hence, TF Record which is a Tensor Flows binary storage format is used.

Step 4: Final Result:

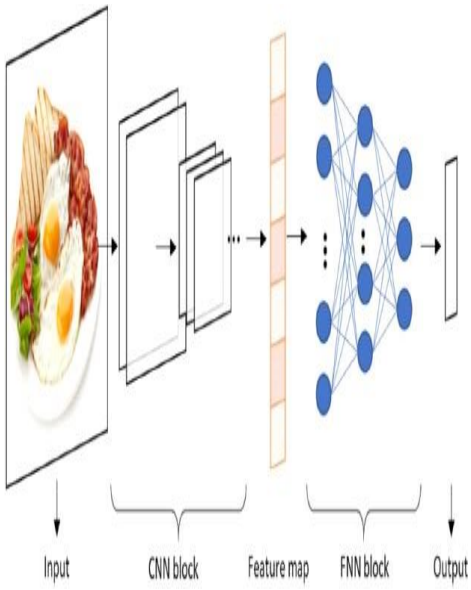
Once the model file is generated from the training, we load it into the Android Application and test it against the images captured and saved by the user. The resultant model can be run directly on mobile device or converted to Tensor Flow Lite format to leverage Android’s Neural Network APIs.

CHAPTER 4: SYSTEM DESIGN

V. SYSTEM DESIGN AND ARCHITECTURE



4.1.1 Block Diagram



CHAPTER 5: SYSTEM IMPLEMENTATION:

5.1 List of Modules

- Image dataset
- Data Pre-Processing
- Classification
- Comparative Performance Analysis

- **IMAGE DATASET**

Dataset is a collection of data. Most commonly a data set corresponds to the contents of a single database table, or a single statistical data matrix, where every column of the table represents a particular variable, and each row corresponds to a given member of the data set in question. The data set lists values for each of the variables, such as height and weight of an object, for each member of the data set. Each value is known as a datum. The data set may comprise data for one or more members, corresponding to the number of rows. The term data set may also be used more loosely, to refer to the data in a collection of closely related tables, corresponding to a particular experiment or event. This collected data stored in the data warehouse.

- **DATA PRE-PROCESSING**

Pre-processing is defined as the removal of error in the data and most important phase of Machine learning project.If there is much irrelevant and redundant information present or

noisy and unreliable data,then knowledge discovery during the training phase is more difficult.Data preparation and filtering steps can take considerable amount of processing time.Data preprocessing includes cleaning, selection, normalization, transformation, feature extraction and selection, etc. Data cleansing or data cleaning is the process of detecting and correcting corrupt or inaccurate records from a records from a record set, table, or database and refers to identifying incomplete, incorrect, inaccurate or irrelevant parts of the data and then replacing, modifying, or deleting the dirty or coarse data. Instance selection is an important data pre-processing step that can be applied in many machine learning tasks. It can be applied for reducing the original dataset to a manageable volume, leading to a reduction of the computational resources that are necessary for performing the learning process. Database normalization is the process of organizing the fields and tables of a relational database to minimize redundancy and dependency. Data transformation is the process of transform raw data and un-structured data into a structured data. Data transformation is typically performed via a mixture of manual and automated steps. Tools and technologies used for data transformation can vary widely based on the format, structure, complexity, and volume of the data being transformed. Source data is collected and stored in data warehouse and is pre-processed to extract the consistent data.

TYPES OF PREPROCESSING:

RGB image, Gray scale image, Binary image

RGB COLOR MODEL

The RGB color model is an additive color model in which red, green and blue light are added together in various ways to reproduce a broad array of colors. The main purpose of the RGB color model is for the sensing, representation and display of images in electronic systems, such as televisions and computers, though it has also been used in conventional photography.

GRAY SCALE MODEL

Gray scale transformations can be performed using look-up tables. Grey scale transformations are mostly used if the result viewed by a human. Grey scale transformations do not depend on the position of the pixel in the image.

BINARY IMAGE MODEL

A binary image is a digital image that has only two possible values for each pixel. Typically, the two colors used for a binary image are black and white. The color used for the object, in the image is the foreground color while the rest of the image is the background color. In the document-scanning industry, this is often referred to as "bi-tonal".

Binary images are also called bi-level or two-level. This means that each pixel is stored as a single bit (i.e., a 0 or 1). The names black-and-white, B&W, monochrome or monochromatic are often used for this concept, designate any images that have only one sample per pixel, such as grayscale images.

PROCESSED DATA

The process results in converting the data from un-structured format to structured form of data which imports the classification of breast cancer.

- **CLASSIFICATION**

Classification is a general process related to categorization, the process in which ideas and objects are recognized, differentiated, and understood. In this case, the features are collected and classified by two various methods.

1. Feature based classification
2.Convolutional Neural Networks (CNN)

CONVOLUTIONAL NEURAL NETWORKS (CNN)

In machine learning, Convolutional Neural Networks is a class of deep networks, and CNN were inspired by biological processes. The connectivity pattern between neurons and resembles the organization of the animal visual cortex. A CNN consists of an input and an output layer, as well as multiple hidden layers. The hidden layers of a CNN typically consist of

Convolutional layers
Pooling layers
Normalization layers
Fully connected
layers

CONVOLUTIONAL LAYERS

The first layer in a CNN is always a Convolutional Layer. Convolutional layers apply a convolution operation to the input, passing the result to the next layer. The convolution emulates the response of an individual neuron to visual stimuli. Each convolutional neuron processes data only for its receptive field. To learn features as well as classify data fully connected feedforward neural networks can be used.

POOLING LAYERS

Convolutional networks may include local or global pooling layers, which combine the outputs of neuron clusters at one layer into a single neuron in the next layer. For example, max pooling uses the maximum value from each of a cluster of neurons at the prior layer. Another example is average pooling, which uses the average value from each of a cluster of neurons at the prior layer.

FULLY CONNECTED LAYERS

Finally, after several convolutional and max pooling layers, the high-level reasoning in the neural network is done via fully connected layers. Neurons in a fully connected layer have connections to all activations in the previous layer, as seen in regular neural networks. Their activations can hence be computed with a matrix multiplication followed by a bias offset.

- **COMPARATIVE PERFORMANCE ANALYSIS**

Precision - Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. The question that this metric answer is of all passed, how manyactually passed High precision relates to the low false positive rate.

PRECISION = TP/TP+FP

Precision is used with recall, the percent of all relevant documents that is returned by the search. The two measures are sometimes used together in the F1 Score (or f-measure) to provide a single measurement for a system. The usage of "precision" in the field of information retrieval differs from the definition of accuracy and precision within other branches of science and technology.

Recall (Sensitivity) - Recall is the ratio of correctly predicted positive observations to the all observations in actual class - yes.

RECALL = TP/TP+FN

For example, for a text search on a set of documents, recall is the number of correct results divided by the number of results that should have been returned. It can be viewed as the probability that a relevant document is retrieved by the query.It is trivial to achieve recall of 100% by returning all documents in response to any query. Therefore, recall alone is not enough but one needs to measure the number of non-relevant documents also, for example by also computing the precision.

F1 score - F1 Score is the weighted average of Precision and Recall. Therefore, this score takes both false positives and false negatives into account. Intuitively it is not as easy to understand as accuracy, but F1 is usually more useful than accuracy, especially if you have an uneven class distribution. Accuracy works best if false positives and false negatives have similar cost. If the cost of false positives and false negatives are very different, it’s better to look at both Precision and Recall.

F1 SCORE = 2*(RECALL * PRECISION) / (RECALL + PRECISION)

This measure is approximately the average of the two when they are close,and is more generally the harmonic mean, which, for the case of two numbers, coincides with the square of the geometric mean divided by the arithmetic mean.

Accuracy - Accuracy is the most intuitive performance measure and it is simply a ratio of correctly predicted observation to the total observations. One may think that, if we have high accuracy then our model is best. Yes, accuracy is a great measure but only when you have symmetric datasets where values of false positive and false negatives are almost same. Therefore, you have to look at other parameters to evaluate the performance of your model.

ACCURACY = TP+TN/TP+FP+FN+TN

- **True Positives (TP)** - These are the correctly predicted positive values which means that the value of actual class is yes and the value of predicted class is also yes. E.g. if actual class value indicates that this passenger survived and predicted class tells you the same thing.

- **True Negatives (TN) -** These are the correctly predicted negative values which means that the value of actual class is no and value of predicted class is also no. E.g. if actual class says this passenger did not survive and predicted class tells you the same thing. False positives and false negatives, these values occur when your actual class contradicts with the predicted class.

- **False Positives (FP) –** When actual class is no and predicted class is yes. E.g. if actual class says this passenger did not survive but predicted class tells you that this passenger will survive.

- **False Negatives (FN) –** When actual class is yes but predicted class in no. E.g. if actual class value indicates that this passenger survived and predicted class tells you that passenger will die.

	Predicted class		
Actual Class		Class = Yes	Class = No
	Class = Yes	True Positive	False Negative
	Class = No	False Positive	True Negative

FIG 5.1.4.1 OBSERVATION OF CLASSIFICATION

CHAPTER 6: CONCLUSION AND FUTURE ENHANCEMENT:

IX. CONCLUSION

Our aim in this paper is to empower the user by a convenient, intelligent and accurate system that helps them become sensible about their calorie intake. We employed a rather unique combination of Convolutional neural networks and fully connected neural network. We showed that the combination of those two methods provides a powerful instrument to attain a accuracy of food recognition in our system. Our plan for future work is to increase our database of images and send notification to user smartphones regarding the calorie.

VIII. FUTURE ENHANCEMENT

In this system, we provide the details of calorie, carbs, fats and proteins in each food captured by the user. In future we plan to enhance our system by notifying the user about how much the user should intake calorie, carbs, fats and proteins to maintain a healthy diet.

APPENDIX A

SAMPLE CODING:
ButtonBold.java:

```
package ai.fooz.foodanalysis.customfont;

import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.Button;



public class ButtonBold extends Button {
    public ButtonBold(Context context) {
        super(context);
        setassets(context);
    }

    public ButtonBold(Context context, AttributeSet attrs) {
        super(context, attrs);
        setassets(context);
    }
```

```
    public ButtonBold(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    public ButtonBold(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    private void setassets(Context context) {
        setTypeface(Typeface.createFromAsset(context.getAssets(),
"Font/Lato-Semibold.ttf"));



    }
}
```

ButtonRegular.java

```
package ai.fooz.foodanalysis.customfont;

import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.Button;



public class ButtonRegular extends Button {
    public ButtonRegular(Context context) {
        super(context);
        setassets(context);
    }

    public ButtonRegular(Context context, AttributeSet attrs) {
        super(context, attrs);
        setassets(context);
    }

    public ButtonRegular(Context context, AttributeSet attrs, int defStyleAttr) {
```

```
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    public ButtonRegular(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    private void setassets(Context context) {
        setTypeface(Typeface.createFromAsset(context.getAssets(), "Font/Lato-Regular.ttf"));




    }
}
```

EditTextBold.java

```
package ai.fooz.foodanalysis.customfont;

import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.EditText;



public class EditTextBold extends EditText {
    public EditTextBold(Context context) {
        super(context);
        setassets(context);
    }

    public EditTextBold(Context context, AttributeSet attrs) {
        super(context, attrs);
        setassets(context);
    }

    public EditTextBold(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }
```

```
    public EditTextBold(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    private void setassets(Context context) {
        setTypeface(Typeface.createFromAsset(context.getAssets(),
"Font/Lato-Semibold.ttf"));



    }
}
```

EditTextRegular.java

```
package ai.fooz.foodanalysis.customfont;

import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.EditText;


public class EditTextRegular extends EditText {
    public EditTextRegular(Context context) {
        super(context);
        setassets(context);
    }

    public EditTextRegular(Context context, AttributeSet attrs) {
        super(context, attrs);
        setassets(context);
    }

    public EditTextRegular(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    public EditTextRegular(Context context, AttributeSet attrs, int defStyleAttr, int defStyleRes)
{
        super(context, attrs, defStyleAttr);
        setassets(context);
    }
```

```
    private void setassets(Context context) {
        setTypeface(Typeface.createFromAsset(context.getAssets(),
"Font/Lato-Regular.ttf"));



    }
}
```

Textview_Regular.java

```
package ai.fooz.foodanalysis.customfont;

import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.TextView;



public class TextView_Regular extends TextView {
    public TextView_Regular(Context context) {
        super(context);
        setassets(context);
    }

    public TextView_Regular(Context context, AttributeSet attrs) {
        super(context, attrs);
        setassets(context);
    }

    public TextView_Regular(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    public TextView_Regular(Context context, AttributeSet attrs, int defStyleAttr, int
defStyleRes) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }
```

```
    private void setassets(Context context) {
        setTypeface(Typeface.createFromAsset(context.getAssets(),
"Font/Lato-Regular.ttf"));



    }
}
```

Textview SemiBold.java

```
package ai.fooz.foodanalysis.customfont;

import android.content.Context;
import android.graphics.Typeface;
import android.util.AttributeSet;
import android.widget.TextView;



public class TextView_SemiBold extends TextView {
    public TextView_SemiBold(Context context) {
        super(context);
        setassets(context);
    }

    public TextView_SemiBold(Context context, AttributeSet attrs) {
        super(context, attrs);
        setassets(context);
    }

    public TextView_SemiBold(Context context, AttributeSet attrs, int defStyleAttr) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    public TextView_SemiBold(Context context, AttributeSet attrs, int defStyleAttr, int
defStyleRes) {
        super(context, attrs, defStyleAttr);
        setassets(context);
    }

    private void setassets(Context context) {
```

```
        this.setTypeface(Typeface.createFromAsset(context.getAssets(),
"Font/Lato-Semibold.ttf"));


    }
}
```

BorderedText.java

```
package org.tensorflow.demo.env;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Align;
import android.graphics.Paint.Style;
import android.graphics.Rect;
import android.graphics.Typeface;
import java.util.Vector;

/**
 * A class that encapsulates the tedious bits of rendering legible, bordered text onto a canvas.
 */
public class BorderedText {
  private final Paint interiorPaint;
  private final Paint exteriorPaint;

  private final float textSize;

  /**
   * Creates a left-aligned bordered text object with a white interior, and a black exterior with
   * the specified text size.
   *
   * @param textSize text size in pixels
   */
  public BorderedText(final float textSize) {
    this(Color.WHITE, Color.BLACK, textSize);
  }

  /**
```

```
    * Create a bordered text object with the specified interior and exterior colors, text size and
    * alignment.
    *
    * @param interiorColor the interior text color
    * @param exteriorColor the exterior text color
    * @param textSize text size in pixels
    */
   public BorderedText(final int interiorColor, final int exteriorColor, final float textSize) {
     interiorPaint = new Paint();
     interiorPaint.setTextSize(textSize);
     interiorPaint.setColor(interiorColor);
     interiorPaint.setStyle(Style.FILL);
     interiorPaint.setAntiAlias(false);
     interiorPaint.setAlpha(255);

     exteriorPaint = new Paint();
     exteriorPaint.setTextSize(textSize);
     exteriorPaint.setColor(exteriorColor);
     exteriorPaint.setStyle(Style.FILL_AND_STROKE);
     exteriorPaint.setStrokeWidth(textSize / 8);
     exteriorPaint.setAntiAlias(false);
     exteriorPaint.setAlpha(255);

     this.textSize = textSize;
   }

   public void setTypeface(Typeface typeface) {
     interiorPaint.setTypeface(typeface);
     exteriorPaint.setTypeface(typeface);
   }

   public void drawText(final Canvas canvas, final float posX, final float posY, final String text)
{
     canvas.drawText(text, posX, posY, exteriorPaint);
     canvas.drawText(text, posX, posY, interiorPaint);
   }

   public void drawLines(Canvas canvas, final float posX, final float posY, Vector<String>
lines) {
     int lineNum = 0;
     for (final String line : lines) {
       drawText(canvas, posX, posY - getTextSize() * (lines.size() - lineNum - 1), line);
```

```
        ++lineNum;
      }
    }

    public void setInteriorColor(final int color) {
      interiorPaint.setColor(color);
    }

    public void setExteriorColor(final int color) {
      exteriorPaint.setColor(color);
    }

    public float getTextSize() {
      return textSize;
    }

    public void setAlpha(final int alpha) {
      interiorPaint.setAlpha(alpha);
      exteriorPaint.setAlpha(alpha);
    }

    public void getTextBounds(
        final String line, final int index, final int count, final Rect lineBounds) {
      interiorPaint.getTextBounds(line, index, count, lineBounds);
    }

    public void setTextAlign(final Align align) {
      interiorPaint.setTextAlign(align);
      exteriorPaint.setTextAlign(align);
    }
}
```

Classifier.java

```
package ai.fooz.foodanalysis.env;

import android.graphics.Bitmap;
import android.graphics.RectF;
import java.util.List;

/**
 * Generic interface for interacting with different recognition engines.
```

```
 */
public interface Classifier {
  /**
   * An immutable result returned by a Classifier describing what was recognized.
   */
  public class Recognition {
    /**
     * A unique identifier for what has been recognized. Specific to the class, not the instance of
     * the object.
     */
    private final String id;

    /**
     * Display name for the recognition.
     */
    private final String title;

    /**
     * A sortable score for how good the recognition is relative to others. Higher should be better.
     */
    private final Float confidence;

    /** Optional location within the source image for the location of the recognized object. */
    private RectF location;

    public Recognition(
        final String id, final String title, final Float confidence, final RectF location) {
      this.id = id;
      this.title = title;
      this.confidence = confidence;
      this.location = location;
    }

    public String getId() {
      return id;
    }

    public String getTitle() {
      return title;
    }

    public Float getConfidence() {
```

```
      return confidence;
    }

    public RectF getLocation() {
      return new RectF(location);
    }

    public void setLocation(RectF location) {
      this.location = location;
    }

    @Override
    public String toString() {
      String resultString = "";
      if (id != null) {
        resultString += "[" + id + "] ";
      }

      if (title != null) {
        resultString += title + " ";
      }

      if (confidence != null) {
        resultString += String.format("(%.1f%%) ", confidence * 100.0f);
      }

      if (location != null) {
        resultString += location + " ";
      }

      return resultString.trim();
    }
  }

  List<Recognition> recognizeImage(Bitmap bitmap);

  void enableStatLogging(final boolean debug);

  String getStatString();

  void close();
}
```

ClassifierActivity.java

```
package ai.fooz.foodanalysis.env;

import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.Typeface;
import android.media.Image;
import android.media.Image.Plane;
import android.media.ImageReader;
import android.media.ImageReader.OnImageAvailableListener;
import android.os.Trace;
import android.util.Size;
import android.util.TypedValue;
import android.view.Display;

import java.util.Vector;
import org.tensorflow.demo.env.BorderedText;
import org.tensorflow.demo.env.ImageUtils;
import org.tensorflow.demo.env.Logger;

import ai.fooz.foodanalysis.CameraActivity;

public class ClassifierActivity extends CameraActivity implements OnImageAvailableListener
{
  private static final Logger LOGGER = new Logger();

  // These are the settings for the original v1 Inception model. If you want to
  // use a model that's been produced from the TensorFlow for Poets codelab,
  // you'll need to set IMAGE_SIZE = 299, IMAGE_MEAN = 128, IMAGE_STD = 128,
  // INPUT_NAME = "Mul", and OUTPUT_NAME = "final_result".
  // You'll also need to update the MODEL_FILE and LABEL_FILE paths to point to
  // the ones you produced.
  //
  // To use v3 Inception model, strip the DecodeJpeg Op from your retrained
  // model first:
  //
  // python strip_unused.py \
```

```
// --input_graph=<retrained-pb-file> \
// --output_graph=<your-stripped-pb-file> \
// --input_node_names="Mul" \
// --output_node_names="final_result" \
// --input_binary=true

/* Inception V3
private static final int INPUT_SIZE = 299;
private static final int IMAGE_MEAN = 128;
private static final float IMAGE_STD = 128.0f;
private static final String INPUT_NAME = "Mul:0";
private static final String OUTPUT_NAME = "final_result";
*/

private static final int INPUT_SIZE = 224;
private static final int IMAGE_MEAN = 128;
private static final float IMAGE_STD = 128.0f;
private static final String INPUT_NAME = "input";
private static final String OUTPUT_NAME = "final_result";

private static final String MODEL_FILE = "file:///android_asset/graph.pb";
private static final String LABEL_FILE = "file:///android_asset/labels.txt";

private static final boolean SAVE_PREVIEW_BITMAP = false;

private static final boolean MAINTAIN_ASPECT = true;

private static final Size DESIRED_PREVIEW_SIZE = new Size(640, 480);

private Classifier classifier;

private Integer sensorOrientation;

private int previewWidth = 0;
private int previewHeight = 0;
private byte[][] yuvBytes;
private int[] rgbBytes = null;
private Bitmap rgbFrameBitmap = null;
private Bitmap croppedBitmap = null;

private Bitmap cropCopyBitmap;
```

```
  private boolean computing = false;

  private Matrix frameToCropTransform;
  private Matrix cropToFrameTransform;

// private ResultsView resultsView;

  private BorderedText borderedText;

  private long lastProcessingTimeMs;

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

// @Override
  protected Size getDesiredPreviewFrameSize() {
    return DESIRED_PREVIEW_SIZE;
  }

  private static final float TEXT_SIZE_DIP = 10;

// @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);

    classifier =
        TensorFlowImageClassifier.create(
            getAssets(),
            MODEL_FILE,
            LABEL_FILE,
            INPUT_SIZE,
            IMAGE_MEAN,
            IMAGE_STD,
            INPUT_NAME,
            OUTPUT_NAME);
```

```
//     resultsView = (ResultsView) findViewById(R.id.results);
      previewWidth = size.getWidth();
      previewHeight = size.getHeight();

      final Display display = getWindowManager().getDefaultDisplay();
      final int screenOrientation = display.getRotation();

      LOGGER.i("Sensor orientation: %d, Screen orientation: %d", rotation,
screenOrientation);

      sensorOrientation = rotation + screenOrientation;

      LOGGER.i("Initializing at size %dx%d", previewWidth, previewHeight);
      rgbBytes = new int[previewWidth * previewHeight];
      rgbFrameBitmap = Bitmap.createBitmap(previewWidth, previewHeight,
Config.ARGB_8888);
      croppedBitmap = Bitmap.createBitmap(INPUT_SIZE, INPUT_SIZE,
Config.ARGB_8888);

      frameToCropTransform =
          ImageUtils.getTransformationMatrix(
              previewWidth, previewHeight,
              INPUT_SIZE, INPUT_SIZE,
              sensorOrientation, MAINTAIN_ASPECT);

      cropToFrameTransform = new Matrix();
      frameToCropTransform.invert(cropToFrameTransform);

      yuvBytes = new byte[3][];

//     addCallback(
//         new DrawCallback() {
//           @Override
//           public void drawCallback(final Canvas canvas) {
//             renderDebug(canvas);
//           }
//         });
    }

    @Override
    public void onImageAvailable(final ImageReader reader) {
```

```
    Image image = null;

    try {
      image = reader.acquireLatestImage();

      if (image == null) {
        return;
      }

      if (computing) {
        image.close();
        return;
      }
      computing = true;

      Trace.beginSection("imageAvailable");

      final Plane[] planes =image.getPlanes();
//       fillBytes(planes, yuvBytes);

      final int yRowStride = planes[0].getRowStride();
      final int uvRowStride = planes[1].getRowStride();
      final int uvPixelStride = planes[1].getPixelStride();
      ImageUtils.convertYUV420ToARGB8888(
          yuvBytes[0],
          yuvBytes[1],
          yuvBytes[2],
          previewWidth,
          previewHeight,
          yRowStride,
          uvRowStride,
          uvPixelStride,
          rgbBytes);

      image.close();
    } catch (final Exception e) {
      if (image != null) {
        image.close();
      }
      LOGGER.e(e, "Exception!");
      Trace.endSection();
      return;
```

```
    }

    rgbFrameBitmap.setPixels(rgbBytes, 0, previewWidth, 0, 0, previewWidth,
previewHeight);
    final Canvas canvas = new Canvas(croppedBitmap);
    canvas.drawBitmap(rgbFrameBitmap, frameToCropTransform, null);

    // For examining the actual TF input.
    if (SAVE_PREVIEW_BITMAP) {
      ImageUtils.saveBitmap(croppedBitmap);
    }

//    runInBackground(
//        new Runnable() {
//          @Override
//          public void run() {
//            final long startTime = SystemClock.uptimeMillis();
//            final List<Classifier.Recognition> results =
classifier.recognizeImage(croppedBitmap);
//            lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;
//
//            cropCopyBitmap = Bitmap.createBitmap(croppedBitmap);
//            resultsView.setResults(results);
//            requestRender();
//            computing = false;
//          }
//        });

    Trace.endSection();
  }

// @Override
  public void onSetDebug(boolean debug) {
    classifier.enableStatLogging(debug);
  }

  private void renderDebug(final Canvas canvas) {
//    if (!isDebug()) {
//      return;
//    }
    final Bitmap copy = cropCopyBitmap;
    if (copy != null) {
```

```
        final Matrix matrix = new Matrix();
        final float scaleFactor = 2;
        matrix.postScale(scaleFactor, scaleFactor);
        matrix.postTranslate(
            canvas.getWidth() - copy.getWidth() * scaleFactor,
            canvas.getHeight() - copy.getHeight() * scaleFactor);
        canvas.drawBitmap(copy, matrix, new Paint());

        final Vector<String> lines = new Vector<String>();
        if (classifier != null) {
          String statString = classifier.getStatString();
          String[] statLines = statString.split("\n");
          for (String line : statLines) {
            lines.add(line);
          }
        }

        lines.add("Frame: " + previewWidth + "x" + previewHeight);
        lines.add("Crop: " + copy.getWidth() + "x" + copy.getHeight());
        lines.add("View: " + canvas.getWidth() + "x" + canvas.getHeight());
        lines.add("Rotation: " + sensorOrientation);
        lines.add("Inference time: " + lastProcessingTimeMs + "ms");

        borderedText.drawLines(canvas, 10, canvas.getHeight() - 10, lines);
      }
    }
}
```

Imageutils.java

```
package org.tensorflow.demo.env;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.media.Image;
import android.os.Environment;

import junit.framework.Assert;

import java.io.File;
import java.io.FileOutputStream;
import java.nio.ByteBuffer;
```

```
/**
 * Utility class for manipulating images.
 **/
public class ImageUtils {
    @SuppressWarnings("unused")
    private static final Logger LOGGER = new Logger();

    // This value is 2 ^ 18 - 1, and is used to clamp the RGB values before their ranges
    // are normalized to eight bits.
    static final int kMaxChannelValue = 262143;


    /**
     * Utility method to compute the allocated size in bytes of a YUV420SP image
     * of the given dimensions.
     */
    public static int getYUVByteSize(final int width, final int height) {
        // The luminance plane requires 1 byte per pixel.
        final int ySize = width * height;

        // The UV plane works on 2x2 blocks, so dimensions with odd size must be rounded up.
        // Each 2x2 block takes 2 bytes to encode, one each for U and V.
        final int uvSize = ((width + 1) / 2) * ((height + 1) / 2) * 2;

        return ySize + uvSize;
    }

    /**
     * Saves a Bitmap object to disk for analysis.
     *
     * @param bitmap The bitmap to save.
     */
    public static void saveBitmap(final Bitmap bitmap) {
        final String root =
                Environment.getExternalStorageDirectory().getAbsolutePath() + File.separator
+ "tensorflow";
        LOGGER.i("Saving %dx%d bitmap to %s.", bitmap.getWidth(), bitmap.getHeight(),
root);
        final File myDir = new File(root);

        if (!myDir.mkdirs()) {
```

```
            LOGGER.i("Make dir failed");
        }

        final String fname = "preview.png";
        final File file = new File(myDir, fname);
        if (file.exists()) {
            file.delete();
        }
        try {
            final FileOutputStream out = new FileOutputStream(file);
            bitmap.compress(Bitmap.CompressFormat.PNG, 99, out);
            out.flush();
            out.close();
        } catch (final Exception e) {
            LOGGER.e(e, "Exception!");
        }
    }


    public static int[] convertImageToBitmap(Image image, int[] output, byte[][]
cachedYuvBytes) {
        if (cachedYuvBytes == null || cachedYuvBytes.length != 3) {
            cachedYuvBytes = new byte[3][];
        }
        Image.Plane[] planes = image.getPlanes();
        fillBytes(planes, cachedYuvBytes);

        final int yRowStride = planes[0].getRowStride();
        final int uvRowStride = planes[1].getRowStride();
        final int uvPixelStride = planes[1].getPixelStride();

        convertYUV420ToARGB8888(cachedYuvBytes[0], cachedYuvBytes[1],
cachedYuvBytes[2],
                image.getWidth(), image.getHeight(), yRowStride, uvRowStride, uvPixelStride,
output);
        return output;
    }

    public static void convertYUV420ToARGB8888(byte[] yData, byte[] uData, byte[] vData,
int width, int height,int yRowStride, int uvRowStride, int uvPixelStride, int[] out) {
        int i = 0;
        for (int y = 0; y < height; y++) {
```

```
            int pY = yRowStride * y;
            int uv_row_start = uvRowStride * (y >> 1);
            int pU = uv_row_start;
            int pV = uv_row_start;

            for (int x = 0; x < width; x++) {
                int uv_offset = (x >> 1) * uvPixelStride;
                out[i++] = YUV2RGB(
                        convertByteToInt(yData, pY + x),
                        convertByteToInt(uData, pU + uv_offset),
                        convertByteToInt(vData, pV + uv_offset));
            }
        }
    }

    private static int convertByteToInt(byte[] arr, int pos) {
        return arr[pos] & 0xFF;
    }

    private static int YUV2RGB(int nY, int nU, int nV) {
        nY -= 16;
        nU -= 128;
        nV -= 128;
        if (nY < 0) nY = 0;

        // This is the floating point equivalent. We do the conversion in integer
        // because some Android devices do not have floating point in hardware.
        // nR = (int)(1.164 * nY + 2.018 * nU);
        // nG = (int)(1.164 * nY - 0.813 * nV - 0.391 * nU);
        // nB = (int)(1.164 * nY + 1.596 * nV);

        int nR = (int) (1192 * nY + 1634 * nV);
        int nG = (int) (1192 * nY - 833 * nV - 400 * nU);
        int nB = (int) (1192 * nY + 2066 * nU);

        nR = Math.min(kMaxChannelValue, Math.max(0, nR));
        nG = Math.min(kMaxChannelValue, Math.max(0, nG));
        nB = Math.min(kMaxChannelValue, Math.max(0, nB));

        nR = (nR >> 10) & 0xff;
        nG = (nG >> 10) & 0xff;
        nB = (nB >> 10) & 0xff;
```

```
        return 0xff000000 | (nR << 16) | (nG << 8) | nB;
    }

    private static void fillBytes(final Image.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 || yuvBytes[i].length != buffer.capacity()) {
                yuvBytes[i] = new byte[buffer.capacity()];
            }
            buffer.get(yuvBytes[i]);
        }
    }


    public static void cropAndRescaleBitmap(final Bitmap src, final Bitmap dst, int
sensorOrientation) {
        Assert.assertEquals(dst.getWidth(), dst.getHeight());
        final float minDim = Math.min(src.getWidth(), src.getHeight());

        final Matrix matrix = new Matrix();

        // We only want the center square out of the original rectangle.
        final float translateX = -Math.max(0, (src.getWidth() - minDim) / 2);
        final float translateY = -Math.max(0, (src.getHeight() - minDim) / 2);
        matrix.preTranslate(translateX, translateY);

        final float scaleFactor = dst.getHeight() / minDim;
        matrix.postScale(scaleFactor, scaleFactor);

        // Rotate around the center if necessary.
        if (sensorOrientation != 0) {
            matrix.postTranslate(-dst.getWidth() / 2.0f, -dst.getHeight() / 2.0f);
            matrix.postRotate(sensorOrientation);
            matrix.postTranslate(dst.getWidth() / 2.0f, dst.getHeight() / 2.0f);
        }

        final Canvas canvas = new Canvas(dst);
        canvas.drawBitmap(src, matrix, null);
    }
```

```
public static Matrix getTransformationMatrix(
        final int srcWidth,
        final int srcHeight,
        final int dstWidth,
        final int dstHeight,
        final int applyRotation,
        final boolean maintainAspectRatio) {
    final Matrix matrix = new Matrix();

    if (applyRotation != 0) {
        // Translate so center of image is at origin.
        matrix.postTranslate(-srcWidth / 2.0f, -srcHeight / 2.0f);

        // Rotate around origin.
        matrix.postRotate(applyRotation);
    }

    // Account for the already applied rotation, if any, and then determine how
    // much scaling is needed for each axis.
    final boolean transpose = (Math.abs(applyRotation) + 90) % 180 == 0;

    final int inWidth = transpose ? srcHeight : srcWidth;
    final int inHeight = transpose ? srcWidth : srcHeight;

    // Apply scaling if necessary.
    if (inWidth != dstWidth || inHeight != dstHeight) {
        final float scaleFactorX = dstWidth / (float) inWidth;
        final float scaleFactorY = dstHeight / (float) inHeight;

        if (maintainAspectRatio) {
            // Scale by minimum factor so that dst is filled completely while
            // maintaining the aspect ratio. Some image may fall off the edge.
            final float scaleFactor = Math.max(scaleFactorX, scaleFactorY);
            matrix.postScale(scaleFactor, scaleFactor);
        } else {
            // Scale exactly to fill dst from src.
            matrix.postScale(scaleFactorX, scaleFactorY);
        }
    }

    if (applyRotation != 0) {
```

```
            // Translate back from origin centered reference to destination frame.
            matrix.postTranslate(dstWidth / 2.0f, dstHeight / 2.0f);
        }

        return matrix;
    }
}
```

Logger.java

```
package org.tensorflow.demo.env;

import android.util.Log;

import java.util.HashSet;
import java.util.Set;

/**
 * Wrapper for the platform log function, allows convenient message prefixing and log disabling.
 */
public final class Logger {
  private static final String DEFAULT_TAG = "tensorflow";
  private static final int DEFAULT_MIN_LOG_LEVEL = Log.DEBUG;

  // Classes to be ignored when examining the stack trace
  private static final Set<String> IGNORED_CLASS_NAMES;

  static {
    IGNORED_CLASS_NAMES = new HashSet<String>(3);
    IGNORED_CLASS_NAMES.add("dalvik.system.VMStack");
    IGNORED_CLASS_NAMES.add("java.lang.Thread");
    IGNORED_CLASS_NAMES.add(Logger.class.getCanonicalName());
  }

  private final String tag;
  private final String messagePrefix;
  private int minLogLevel = DEFAULT_MIN_LOG_LEVEL;

  /**
   * Creates a Logger using the class name as the message prefix.
   *
   * @param clazz the simple name of this class is used as the message prefix.
   */
```

```
  public Logger(final Class<?> clazz) {
    this(clazz.getSimpleName());
  }

  /**
   * Creates a Logger using the specified message prefix.
   *
   * @param messagePrefix is prepended to the text of every message.
   */
  public Logger(final String messagePrefix) {
    this(DEFAULT_TAG, messagePrefix);
  }

  /**
   * Creates a Logger with a custom tag and a custom message prefix. If the message prefix
   * is set to <pre>null</pre>, the caller's class name is used as the prefix.
   *
   * @param tag identifies the source of a log message.
   * @param messagePrefix prepended to every message if non-null. If null, the name of the
caller is
   *                      being used
   */
  public Logger(final String tag, final String messagePrefix) {
    this.tag = tag;
    final String prefix = messagePrefix == null ? getCallerSimpleName() : messagePrefix;
    this.messagePrefix = (prefix.length() > 0) ? prefix + ": " : prefix;
  }

  /**
   * Creates a Logger using the caller's class name as the message prefix.
   */
  public Logger() {
    this(DEFAULT_TAG, null);
  }

  /**
   * Creates a Logger using the caller's class name as the message prefix.
   */
  public Logger(final int minLogLevel) {
    this(DEFAULT_TAG, null);
    this.minLogLevel = minLogLevel;
  }
```

```
public void setMinLogLevel(final int minLogLevel) {
   this.minLogLevel = minLogLevel;
}

public boolean isLoggable(final int logLevel) {
   return logLevel >= minLogLevel || Log.isLoggable(tag, logLevel);
}

/**
 * Return caller's simple name.
 *
 * Android getStackTrace() returns an array that looks like this:
 *        stackTrace[0]: dalvik.system.VMStack
 *        stackTrace[1]: java.lang.Thread
 *        stackTrace[2]: com.google.android.apps.unveil.env.UnveilLogger
 *        stackTrace[3]: com.google.android.apps.unveil.BaseApplication
 *
 * This function returns the simple version of the first non-filtered name.
 *
 * @return caller's simple name
 */
private static String getCallerSimpleName() {
   // Get the current callstack so we can pull the class of the caller off of it.
   final StackTraceElement[] stackTrace = Thread.currentThread().getStackTrace();

   for (final StackTraceElement elem : stackTrace) {
      final String className = elem.getClassName();
      if (!IGNORED_CLASS_NAMES.contains(className)) {
         // We're only interested in the simple name of the class, not the complete package.
         final String[] classParts = className.split("\\.");
         return classParts[classParts.length - 1];
      }
   }

   return Logger.class.getSimpleName();
}

private String toMessage(final String format, final Object... args) {
   return messagePrefix + (args.length > 0 ? String.format(format, args) : format);
}
```

```
public void v(final String format, final Object... args) {
  if (isLoggable(Log.VERBOSE)) {
    Log.v(tag, toMessage(format, args));
  }
}

public void v(final Throwable t, final String format, final Object... args) {
  if (isLoggable(Log.VERBOSE)) {
    Log.v(tag, toMessage(format, args), t);
  }
}

public void d(final String format, final Object... args) {
  if (isLoggable(Log.DEBUG)) {
    Log.d(tag, toMessage(format, args));
  }
}

public void d(final Throwable t, final String format, final Object... args) {
  if (isLoggable(Log.DEBUG)) {
    Log.d(tag, toMessage(format, args), t);
  }
}

public void i(final String format, final Object... args) {
  if (isLoggable(Log.INFO)) {
    Log.i(tag, toMessage(format, args));
  }
}

public void i(final Throwable t, final String format, final Object... args) {
  if (isLoggable(Log.INFO)) {
    Log.i(tag, toMessage(format, args), t);
  }
}

public void w(final String format, final Object... args) {
  if (isLoggable(Log.WARN)) {
    Log.w(tag, toMessage(format, args));
  }
}
```

```
  public void w(final Throwable t, final String format, final Object... args) {
    if (isLoggable(Log.WARN)) {
      Log.w(tag, toMessage(format, args), t);
    }
  }

  public void e(final String format, final Object... args) {
    if (isLoggable(Log.ERROR)) {
      Log.e(tag, toMessage(format, args));
    }
  }

  public void e(final Throwable t, final String format, final Object... args) {
    if (isLoggable(Log.ERROR)) {
      Log.e(tag, toMessage(format, args), t);
    }
  }
}
```

MyUtility.java

```
package ai.fooz.foodanalysis.env;

import android.content.Context;
importandroid.content.DialogInterface;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.view.View;

import com.google.firebase.analytics.FirebaseAnalytics;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;

import ai.fooz.foodanalysis.MainActivity;

public class MyUtility {

    private static final String LABEL_FILE = "file:///android_asset/labels.txt";
```

```
public static final float PREDICTION_THRESHOLD = (float) 0.60;

public static String getDateWithFormat(Date dt, String format) {
    SimpleDateFormat df = new SimpleDateFormat(format);
    String formattedDate = df.format(dt);
    return formattedDate;
}

public static String toTitleCase(String str) {

    if (str == null) {
        return null;
    }

    boolean space = true;
    StringBuilder builder = new StringBuilder(str);
    final int len = builder.length();

    for (int i = 0; i < len; ++i) {
        char c = builder.charAt(i);
        if (space) {
            if (!Character.isWhitespace(c)) {
                // Convert to title case and switch out of whitespace mode.
                builder.setCharAt(i, Character.toTitleCase(c));
                space = false;
            }
        } else if (Character.isWhitespace(c)) {
            space = true;
        } else {
            builder.setCharAt(i, Character.toLowerCase(c));
        }
    }

    return builder.toString();
}

public static int getRandomNumber(int min, int max) {
    Random r = new Random();
    final int random = r.nextInt((max - min) + 1) + min;
    return random;
}
```

```
    public static String getLabels(Context myContext) {

        String actualFilename = LABEL_FILE.split("file:///android_asset/")[1];
        String lbls = "";
        BufferedReader br = null;
        try {
            br = new BufferedReader(new
InputStreamReader(myContext.getAssets().open(actualFilename)));
            String line;
            while ((line = br.readLine()) != null) {
                lbls = lbls+"\n"+ MyUtility.toTitleCase(line);
            }
            br.close();
            return lbls;
        } catch (IOException e) {
            throw new RuntimeException("Problem reading label file!" , e);
        }
    }

    public static Boolean logFirebaseEvent(FirebaseAnalytics mFirebaseAnalytics, String
itemName) {
        Bundle bundle = new Bundle();
//          bundle.putString(FirebaseAnalytics.Param.ITEM_ID, id);
        bundle.putString(FirebaseAnalytics.Param.ITEM_NAME, itemName);
//          bundle.putString(FirebaseAnalytics.Param.CONTENT_TYPE, "image");
        mFirebaseAnalytics.logEvent(FirebaseAnalytics.Event.SELECT_CONTENT, bundle);
        return true;
    }
}
```

RecyclerTouch Listener.java

```
package ai.fooz.foodanalysis.env;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;

public class RecyclerTouchListener implements RecyclerView.OnItemTouchListener {

private GestureDetector gestureDetector;
private ClickListener clickListener;
```

```
public RecyclerTouchListener(Context context, final RecyclerView recyclerView, final
ClickListener clickListener) {
        this.clickListener = clickListener;
        gestureDetector = new GestureDetector(context, new
GestureDetector.SimpleOnGestureListener() {
@Override
public boolean onSingleTapUp(MotionEvent e) {
        return true;
        }

@Override
public void onLongPress(MotionEvent e) {
        View child = recyclerView.findChildViewUnder(e.getX(), e.getY());
        if (child != null && clickListener != null) {
        clickListener.onLongClick(child, recyclerView.getChildPosition(child));
        }
        }
        });
        }

@Override
public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {

        View child = rv.findChildViewUnder(e.getX(), e.getY());
        if (child != null && clickListener != null && gestureDetector.onTouchEvent(e)) {
        clickListener.onClick(child, rv.getChildPosition(child));
        }
        return false;
        }

@Override
public void onTouchEvent(RecyclerView rv, MotionEvent e) {
        }

@Override
public void onRequestDisallowInterceptTouchEvent(boolean disallowIntercept) {

        }

public interface ClickListener {
    void onClick(View view, int position);
```

```
    void onLongClick(View view, int position);
}
}
```

Size.java

```
package org.tensorflow.demo.env;

import android.graphics.Bitmap;
import android.text.TextUtils;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * Size class independent of a Camera object.
 */
public class Size implements Comparable<Size>, Serializable {

  // 1.4 went out with this UID so we'll need to maintain it to preserve pending queries when
  // upgrading.
  public static final long serialVersionUID = 7689808733290872361L;

  public final int width;
  public final int height;

  public Size(final int width, final int height) {
    this.width = width;
    this.height = height;
  }

  public Size(final Bitmap bmp) {
    this.width = bmp.getWidth();
    this.height = bmp.getHeight();
  }

  /**
   * Rotate a size by the given number of degrees.
   * @param size Size to rotate.
   * @param rotation Degrees {0, 90, 180, 270} to rotate the size.
   * @return Rotated size.
```

```
 */
public static Size getRotatedSize(final Size size, final int rotation) {
  if (rotation % 180 != 0) {
    // The phone is portrait, therefore the camera is sideways and frame should be rotated.
    return new Size(size.height, size.width);
  }
  return size;
}

public static Size parseFromString(String sizeString) {
  if (TextUtils.isEmpty(sizeString)) {
    return null;
  }

  sizeString = sizeString.trim();

  // The expected format is "<width>x<height>".
  final String[] components = sizeString.split("x");
  if (components.length == 2) {
    try {
      final int width = Integer.parseInt(components[0]);
      final int height = Integer.parseInt(components[1]);
      return new Size(width, height);
    } catch (final NumberFormatException e) {
      return null;
    }
  } else {
    return null;
  }
}

public static List<Size> sizeStringToList(final String sizes) {
  final List<Size> sizeList = new ArrayList<Size>();
  if (sizes != null) {
    final String[] pairs = sizes.split(",");
    for (final String pair : pairs) {
      final Size size = Size.parseFromString(pair);
      if (size != null) {
        sizeList.add(size);
      }
    }
  }
```

```
    return sizeList;
  }

  public static String sizeListToString(final List<Size> sizes) {
    String sizesString = "";
    if (sizes != null && sizes.size() > 0) {
      sizesString = sizes.get(0).toString();
      for (int i = 1; i < sizes.size(); i++) {
        sizesString += "," + sizes.get(i).toString();
      }
    }
    return sizesString;
  }

  public final float aspectRatio() {
    return (float) width / (float) height;
  }

  @Override
  public int compareTo(final Size other) {
    return width * height - other.width * other.height;
  }

  @Override
  public boolean equals(final Object other) {
    if (other == null) {
      return false;
    }

    if (!(other instanceof Size)) {
      return false;
    }

    final Size otherSize = (Size) other;
    return (width == otherSize.width && height == otherSize.height);
  }

  @Override
  public int hashCode() {
    return width * 32713 + height;
  }
```

```
  @Override
  public String toString() {
    return dimensionsAsString(width, height);
  }

  public static final String dimensionsAsString(final int width, final int height) {
    return width + "x" + height;
  }
}
```

SplitTimer.java

```
package org.tensorflow.demo.env;

import android.os.SystemClock;

/**
 * A simple utility timer for measuring CPU time and wall-clock splits.
 */
public class SplitTimer {
  private final Logger logger;

  private long lastWallTime;
  private long lastCpuTime;

  public SplitTimer(final String name) {
    logger = new Logger(name);
    newSplit();
  }

  public void newSplit() {
    lastWallTime = SystemClock.uptimeMillis();
    lastCpuTime = SystemClock.currentThreadTimeMillis();
  }

  public void endSplit(final String splitName) {
    final long currWallTime = SystemClock.uptimeMillis();
    final long currCpuTime = SystemClock.currentThreadTimeMillis();

    logger.i(
        "%s: cpu=%dms wall=%dms",
        splitName, currCpuTime - lastCpuTime, currWallTime - lastWallTime);
```

```
      lastWallTime = currWallTime;
      lastCpuTime = currCpuTime;
    }
  }
```

TensorFlowImageClassifier.java

```
package ai.fooz.foodanalysis.env;

import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.os.Trace;
import android.util.Log;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;
import java.util.Vector;
import org.tensorflow.Operation;
import org.tensorflow.contrib.android.TensorFlowInferenceInterface;

/** A classifier specialized to label images using TensorFlow. */
public class TensorFlowImageClassifier implements Classifier {
  public static final String TAG = "TensorFlowImageClassifier";

  // Only return this many results with at least this confidence.
  public static final int MAX_RESULTS = 3;
  public static final float THRESHOLD = 0.1f;

  // Config values.
  public String inputName;
  public String outputName;
  public int inputSize;
  public int imageMean;
  public float imageStd;

  // Pre-allocated buffers.
  public Vector<String> labels = new Vector<String>();
  public int[] intValues;
```

```
public float[] floatValues;
public float[] outputs;
public String[] outputNames;

public boolean logStats = false;

public TensorFlowInferenceInterface inferenceInterface;

public TensorFlowImageClassifier() {}

/**
 * Initializes a native TensorFlow session for classifying images.
 *
 * @param assetManager The asset manager to be used to load assets.
 * @param modelFilename The filepath of the model GraphDef protocol buffer.
 * @param labelFilename The filepath of label file for classes.
 * @param inputSize The input size. A square image of inputSize x inputSize is assumed.
 * @param imageMean The assumed mean of the image values.
 * @param imageStd The assumed std of the image values.
 * @param inputName The label of the image input node.
 * @param outputName The label of the output node.
 * @throws IOException
 */
public static Classifier create(
    AssetManager assetManager,
    String modelFilename,
    String labelFilename,
    int inputSize,
    int imageMean,
    float imageStd,
    String inputName,
    String outputName) {
  TensorFlowImageClassifier c = new TensorFlowImageClassifier();
  c.inputName = inputName;
  c.outputName = outputName;

  // Read the label names into memory.
  // TODO(andrewharp): make this handle non-assets.
  String actualFilename = labelFilename.split("file:///android_asset/")[1];
  Log.i(TAG, "Reading labels from: " + actualFilename);
  BufferedReader br = null;
  try {
```

```
      br = new BufferedReader(new InputStreamReader(assetManager.open(actualFilename)));
      String line;
      while ((line = br.readLine()) != null) {
        c.labels.add(line);
      }
      br.close();
    } catch (IOException e) {
      throw new RuntimeException("Problem reading label file!" , e);
    }

    c.inferenceInterface = new TensorFlowInferenceInterface(assetManager, modelFilename);

    // The shape of the output is [N, NUM_CLASSES], where N is the batch size.
    final Operation operation = c.inferenceInterface.graphOperation(outputName);
    final int numClasses = (int) operation.output(0).shape().size(1);
    Log.i(TAG, "Read " + c.labels.size() + " labels, output layer size is " + numClasses);

    // Ideally, inputSize could have been retrieved from the shape of the input operation.  Alas,
    // the placeholder node for input in the graphdef typically used does not specify a shape, so it
    // must be passed in as a parameter.
    c.inputSize = inputSize;
    c.imageMean = imageMean;
    c.imageStd = imageStd;

    // Pre-allocate buffers.
    c.outputNames = new String[] {outputName};
    c.intValues = new int[inputSize * inputSize];
    c.floatValues = new float[inputSize * inputSize * 3];
    c.outputs = new float[numClasses];

    return c;
  }

  @Override
  public List<Recognition> recognizeImage(final Bitmap bitmap) {
    // Log this method so that it can be analyzed with systrace.
    Trace.beginSection("recognizeImage");

    Trace.beginSection("preprocessBitmap");
    // Preprocess the image data from 0-255 int to normalized float based
    // on the provided parameters.
    bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(),
```

```
bitmap.getHeight());
    for (int i = 0; i < intValues.length; ++i) {
      final int val = intValues[i];
      floatValues[i * 3 + 0] = (((val >> 16) & 0xFF) - imageMean) / imageStd;
      floatValues[i * 3 + 1] = (((val >> 8) & 0xFF) - imageMean) / imageStd;
      floatValues[i * 3 + 2] = ((val & 0xFF) - imageMean) / imageStd;
    }
    Trace.endSection();

    // Copy the input data into TensorFlow.
    Trace.beginSection("feed");
    inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
    Trace.endSection();

    // Run the inference call.
    Trace.beginSection("run");
    inferenceInterface.run(outputNames, logStats);
    Trace.endSection();

    // Copy the output Tensor back into the output array.
    Trace.beginSection("fetch");
    inferenceInterface.fetch(outputName, outputs);
    Trace.endSection();

    // Find the best classifications.
    PriorityQueue<Recognition> pq =
        new PriorityQueue<Recognition>(
            3,
            new Comparator<Recognition>() {
              @Override
              public int compare(Recognition lhs, Recognition rhs) {
                // Intentionally reversed to put high confidence at the head of the queue.
                return Float.compare(rhs.getConfidence(), lhs.getConfidence());
              }
            });
    for (int i = 0; i < outputs.length; ++i) {
      if (outputs[i] > THRESHOLD) {
        pq.add(
            new Recognition(
                "" + i, labels.size() > i ? labels.get(i) : "unknown", outputs[i], null));
      }
    }
```

```
    final ArrayList<Recognition> recognitions = new ArrayList<Recognition>();
    int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
    for (int i = 0; i < recognitionsSize; ++i) {
      recognitions.add(pq.poll());
    }
    Trace.endSection(); // "recognizeImage"
    return recognitions;
  }

  @Override
  public void enableStatLogging(boolean logStats) {
    this.logStats = logStats;
  }

  @Override
  public String getStatString() {
    return inferenceInterface.getStatString();
  }

  @Override
  public void close() {
    inferenceInterface.close();
  }
}
```

BorderedText.java

```
package org.tensorflow.demo.env;

import android.graphics.Canvas;
import android.graphics.Color;
import android.graphics.Paint;
import android.graphics.Paint.Align;
import android.graphics.Paint.Style;
import android.graphics.Rect;
import android.graphics.Typeface;
import java.util.Vector;

/**
 * A class that encapsulates the tedious bits of rendering legible, bordered text onto a canvas.
 */
```

```
public class BorderedText {
  private final Paint interiorPaint;
  private final Paint exteriorPaint;

  private final float textSize;

  /**
   * Creates a left-aligned bordered text object with a white interior, and a black exterior with
   * the specified text size.
   *
   * @param textSize text size in pixels
   */
  public BorderedText(final float textSize) {
    this(Color.WHITE, Color.BLACK, textSize);
  }

  /**
   * Create a bordered text object with the specified interior and exterior colors, text size and
   * alignment.
   *
   * @param interiorColor the interior text color
   * @param exteriorColor the exterior text color
   * @param textSize text size in pixels
   */
  public BorderedText(final int interiorColor, final int exteriorColor, final float textSize) {
    interiorPaint = new Paint();
    interiorPaint.setTextSize(textSize);
    interiorPaint.setColor(interiorColor);
    interiorPaint.setStyle(Style.FILL);
    interiorPaint.setAntiAlias(false);
    interiorPaint.setAlpha(255);

    exteriorPaint = new Paint();
    exteriorPaint.setTextSize(textSize);
    exteriorPaint.setColor(exteriorColor);
    exteriorPaint.setStyle(Style.FILL_AND_STROKE);
    exteriorPaint.setStrokeWidth(textSize / 8);
    exteriorPaint.setAntiAlias(false);
    exteriorPaint.setAlpha(255);

    this.textSize = textSize;
  }
```

```
  public void setTypeface(Typeface typeface) {
    interiorPaint.setTypeface(typeface);
    exteriorPaint.setTypeface(typeface);
  }

  public void drawText(final Canvas canvas, final float posX, final float posY, final String text)
{
    canvas.drawText(text, posX, posY, exteriorPaint);
    canvas.drawText(text, posX, posY, interiorPaint);
  }

  public void drawLines(Canvas canvas, final float posX, final float posY, Vector<String>
lines) {
    int lineNum = 0;
    for (final String line : lines) {
      drawText(canvas, posX, posY - getTextSize() * (lines.size() - lineNum - 1), line);
      ++lineNum;
    }
  }

  public void setInteriorColor(final int color) {
    interiorPaint.setColor(color);
  }

  public void setExteriorColor(final int color) {
    exteriorPaint.setColor(color);
  }

  public float getTextSize() {
    return textSize;
  }

  public void setAlpha(final int alpha) {
    interiorPaint.setAlpha(alpha);
    exteriorPaint.setAlpha(alpha);
  }

  public void getTextBounds(
      final String line, final int index, final int count, final Rect lineBounds) {
    interiorPaint.getTextBounds(line, index, count, lineBounds);
  }
```

```
  public void setTextAlign(final Align align) {
    interiorPaint.setTextAlign(align);
    exteriorPaint.setTextAlign(align);
  }
}
```

Classifier.java

```
package ai.fooz.foodanalysis.env;

import android.graphics.Bitmap;
import android.graphics.RectF;
import java.util.List;

/**
 * Generic interface for interacting with different recognition engines.
 */
public interface Classifier {
  /**
   * An immutable result returned by a Classifier describing what was recognized.
   */
  public class Recognition {
    /**
     * A unique identifier for what has been recognized. Specific to the class, not the instance of
     * the object.
     */
    private final String id;

    /**
     * Display name for the recognition.
     */
    private final String title;

    /**
     * A sortable score for how good the recognition is relative to others. Higher should be better.
     */
    private final Float confidence;

    /** Optional location within the source image for the location of the recognized object. */
    private RectF location;

    public Recognition(
```

```
      final String id, final String title, final Float confidence, final RectF location) {
    this.id = id;
    this.title = title;
    this.confidence = confidence;
    this.location = location;
  }

  public String getId() {
    return id;
  }

  public String getTitle() {
    return title;
  }

  public Float getConfidence() {
    return confidence;
  }

  public RectF getLocation() {
    return new RectF(location);
  }

  public void setLocation(RectF location) {
    this.location = location;
  }

  @Override
  public String toString() {
    String resultString = "";
    if (id != null) {
      resultString += "[" + id + "] ";
    }

    if (title != null) {
      resultString += title + " ";
    }

    if (confidence != null) {
      resultString += String.format("(%.1f%%) ", confidence * 100.0f);
    }
```

```
        if (location != null) {
          resultString += location + " ";
        }

        return resultString.trim();
      }
    }

    List<Recognition> recognizeImage(Bitmap bitmap);

    void enableStatLogging(final boolean debug);

    String getStatString();

    void close();
}
```

ClassifierActivity.java

```
package ai.fooz.foodanalysis.env;

import android.graphics.Bitmap;
import android.graphics.Bitmap.Config;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.graphics.Paint;
import android.graphics.Typeface;
import android.media.Image;
import android.media.Image.Plane;
import android.media.ImageReader;
import android.media.ImageReader.OnImageAvailableListener;
import android.os.Trace;
import android.util.Size;
import android.util.TypedValue;
import android.view.Display;

import java.util.Vector;
import org.tensorflow.demo.env.BorderedText;
import org.tensorflow.demo.env.ImageUtils;
import org.tensorflow.demo.env.Logger;
```

```
import ai.fooz.foodanalysis.CameraActivity;

public class ClassifierActivity extends CameraActivity implements OnImageAvailableListener
{
  private static final Logger LOGGER = new Logger();

  // These are the settings for the original v1 Inception model. If you want to
  // use a model that's been produced from the TensorFlow for Poets codelab,
  // you'll need to set IMAGE_SIZE = 299, IMAGE_MEAN = 128, IMAGE_STD = 128,
  // INPUT_NAME = "Mul", and OUTPUT_NAME = "final_result".
  // You'll also need to update the MODEL_FILE and LABEL_FILE paths to point to
  // the ones you produced.
  //
  // To use v3 Inception model, strip the DecodeJpeg Op from your retrained
  // model first:
  //
  // python strip_unused.py \
  // --input_graph=<retrained-pb-file> \
  // --output_graph=<your-stripped-pb-file> \
  // --input_node_names="Mul" \
  // --output_node_names="final_result" \
  // --input_binary=true

  /* Inception V3
  private static final int INPUT_SIZE = 299;
  private static final int IMAGE_MEAN = 128;
  private static final float IMAGE_STD = 128.0f;
  private static final String INPUT_NAME = "Mul:0";
  private static final String OUTPUT_NAME = "final_result";
  */

  private static final int INPUT_SIZE = 224;
  private static final int IMAGE_MEAN = 128;
  private static final float IMAGE_STD = 128.0f;
  private static final String INPUT_NAME = "input";
  private static final String OUTPUT_NAME = "final_result";

  private static final String MODEL_FILE = "file:///android_asset/graph.pb";
  private static final String LABEL_FILE = "file:///android_asset/labels.txt";

  private static final boolean SAVE_PREVIEW_BITMAP = false;
```

```
  private static final boolean MAINTAIN_ASPECT = true;

  private static final Size DESIRED_PREVIEW_SIZE = new Size(640, 480);

  private Classifier classifier;

  private Integer sensorOrientation;

  private int previewWidth = 0;
  private int previewHeight = 0;
  private byte[][] yuvBytes;
  private int[] rgbBytes = null;
  private Bitmap rgbFrameBitmap = null;
  private Bitmap croppedBitmap = null;

  private Bitmap cropCopyBitmap;

  private boolean computing = false;

  private Matrix frameToCropTransform;
  private Matrix cropToFrameTransform;

// private ResultsView resultsView;

  private BorderedText borderedText;

  private long lastProcessingTimeMs;

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

// @Override
  protected Size getDesiredPreviewFrameSize() {
    return DESIRED_PREVIEW_SIZE;
  }

  private static final float TEXT_SIZE_DIP = 10;

// @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);

    classifier =
        TensorFlowImageClassifier.create(
            getAssets(),
            MODEL_FILE,
            LABEL_FILE,
            INPUT_SIZE,
            IMAGE_MEAN,
            IMAGE_STD,
            INPUT_NAME,
            OUTPUT_NAME);

//    resultsView = (ResultsView) findViewById(R.id.results);
    previewWidth = size.getWidth();
    previewHeight = size.getHeight();

    final Display display = getWindowManager().getDefaultDisplay();
    final int screenOrientation = display.getRotation();

    LOGGER.i("Sensor orientation: %d, Screen orientation: %d", rotation,
screenOrientation);

    sensorOrientation = rotation + screenOrientation;

    LOGGER.i("Initializing at size %dx%d", previewWidth, previewHeight);
    rgbBytes = new int[previewWidth * previewHeight];
    rgbFrameBitmap = Bitmap.createBitmap(previewWidth, previewHeight,
Config.ARGB_8888);
    croppedBitmap = Bitmap.createBitmap(INPUT_SIZE, INPUT_SIZE,
Config.ARGB_8888);

    frameToCropTransform =
        ImageUtils.getTransformationMatrix(
            previewWidth, previewHeight,
            INPUT_SIZE, INPUT_SIZE,
```

```
            sensorOrientation, MAINTAIN_ASPECT);

    cropToFrameTransform = new Matrix();
    frameToCropTransform.invert(cropToFrameTransform);

    yuvBytes = new byte[3][];

//    addCallback(
//        new DrawCallback() {
//          @Override
//          public void drawCallback(final Canvas canvas) {
//            renderDebug(canvas);
//          }
//        });
  }

  @Override
  public void onImageAvailable(final ImageReader reader) {
    Image image = null;

    try {
      image = reader.acquireLatestImage();

      if (image == null) {
        return;
      }

      if (computing) {
        image.close();
        return;
      }
      computing = true;

      Trace.beginSection("imageAvailable");

      final Plane[] planes = image.getPlanes();
//      fillBytes(planes, yuvBytes);

      final int yRowStride = planes[0].getRowStride();
      final int uvRowStride = planes[1].getRowStride();
      final int uvPixelStride = planes[1].getPixelStride();
      ImageUtils.convertYUV420ToARGB8888(
```

```
            yuvBytes[0],
            yuvBytes[1],
            yuvBytes[2],
            previewWidth,
            previewHeight,
            yRowStride,
            uvRowStride,
            uvPixelStride,
            rgbBytes);

        image.close();
      } catch (final Exception e) {
        if (image != null) {
          image.close();
        }
        LOGGER.e(e, "Exception!");
        Trace.endSection();
        return;
      }

      rgbFrameBitmap.setPixels(rgbBytes, 0, previewWidth, 0, 0, previewWidth,
previewHeight);
      final Canvas canvas = new Canvas(croppedBitmap);
      canvas.drawBitmap(rgbFrameBitmap, frameToCropTransform, null);

      // For examining the actual TF input.
      if (SAVE_PREVIEW_BITMAP) {
        ImageUtils.saveBitmap(croppedBitmap);
      }

//      runInBackground(
//          new Runnable() {
//            @Override
//            public void run() {
//              final long startTime = SystemClock.uptimeMillis();
//              final List<Classifier.Recognition> results =
classifier.recognizeImage(croppedBitmap);
//              lastProcessingTimeMs = SystemClock.uptimeMillis() - startTime;
//
//              cropCopyBitmap = Bitmap.createBitmap(croppedBitmap);
//              resultsView.setResults(results);
//              requestRender();
```

```
//                computing = false;
//              }
//            });

      Trace.endSection();
    }

// @Override
    public void onSetDebug(boolean debug) {
      classifier.enableStatLogging(debug);
    }

    private void renderDebug(final Canvas canvas) {
//      if (!isDebug()) {
//        return;
//      }
      final Bitmap copy = cropCopyBitmap;
      if (copy != null) {
        final Matrix matrix = new Matrix();
        final float scaleFactor = 2;
        matrix.postScale(scaleFactor, scaleFactor);
        matrix.postTranslate(
            canvas.getWidth() - copy.getWidth() * scaleFactor,
            canvas.getHeight() - copy.getHeight() * scaleFactor);
        canvas.drawBitmap(copy, matrix, new Paint());

        final Vector<String> lines = new Vector<String>();
        if (classifier != null) {
          String statString = classifier.getStatString();
          String[] statLines = statString.split("\n");
          for (String line : statLines) {
            lines.add(line);
          }
        }

        lines.add("Frame: " + previewWidth + "x" + previewHeight);
        lines.add("Crop: " + copy.getWidth() + "x" + copy.getHeight());
        lines.add("View: " + canvas.getWidth() + "x" + canvas.getHeight());
        lines.add("Rotation: " + sensorOrientation);
        lines.add("Inference time: " + lastProcessingTimeMs + "ms");

        borderedText.drawLines(canvas, 10, canvas.getHeight() - 10, lines);
```

```
    }
  }
}
```

Imageutils.java

```
package org.tensorflow.demo.env;

import android.graphics.Bitmap;
import android.graphics.Canvas;
import android.graphics.Matrix;
import android.media.Image;
import android.os.Environment;

import junit.framework.Assert;

import java.io.File;
import java.io.FileOutputStream;
import java.nio.ByteBuffer;

/**
 * Utility class for manipulating images.
 **/
public class ImageUtils {
    @SuppressWarnings("unused")
    private static final Logger LOGGER = new Logger();

    // This value is 2 ^ 18 - 1, and is used to clamp the RGB values before their ranges
    // are normalized to eight bits.
    static final int kMaxChannelValue = 262143;


    /**
     * Utility method to compute the allocated size in bytes of a YUV420SP image
     * of the given dimensions.
     */
    public static int getYUVByteSize(final int width, final int height) {
        // The luminance plane requires 1 byte per pixel.
        final int ySize = width * height;

        // The UV plane works on 2x2 blocks, so dimensions with odd size must be rounded up.
        // Each 2x2 block takes 2 bytes to encode, one each for U and V.
        final int uvSize = ((width + 1) / 2) * ((height + 1) / 2) * 2;
```

```
        return ySize + uvSize;
    }

    /**
     * Saves a Bitmap object to disk for analysis.
     *
     * @param bitmap The bitmap to save.
     */
    public static void saveBitmap(final Bitmap bitmap) {
        final String root =
                Environment.getExternalStorageDirectory().getAbsolutePath() + File.separator
+ "tensorflow";
        LOGGER.i("Saving %dx%d bitmap to %s.", bitmap.getWidth(), bitmap.getHeight(),
root);
        final File myDir = new File(root);

        if (!myDir.mkdirs()) {
            LOGGER.i("Make dir failed");
        }

        final String fname = "preview.png";
        final File file = new File(myDir, fname);
        if (file.exists()) {
            file.delete();
        }
        try {
            final FileOutputStream out = new FileOutputStream(file);
            bitmap.compress(Bitmap.CompressFormat.PNG, 99,out);
            out.flush();
            out.close();
        } catch (final Exception e) {
            LOGGER.e(e, "Exception!");
        }
    }


    public static int[] convertImageToBitmap(Image image, int[] output, byte[][]
cachedYuvBytes) {
        if (cachedYuvBytes == null || cachedYuvBytes.length != 3) {
            cachedYuvBytes = new byte[3][];
        }
        Image.Plane[] planes = image.getPlanes();
```

```
        fillBytes(planes, cachedYuvBytes);

        final int yRowStride = planes[0].getRowStride();
        final int uvRowStride = planes[1].getRowStride();
        final int uvPixelStride = planes[1].getPixelStride();

        convertYUV420ToARGB8888(cachedYuvBytes[0], cachedYuvBytes[1],
cachedYuvBytes[2],
                image.getWidth(), image.getHeight(), yRowStride, uvRowStride, uvPixelStride,
output);
        return output;
    }

    public static void convertYUV420ToARGB8888(byte[] yData, byte[] uData, byte[] vData,
int width, int height,
                                               int yRowStride, int uvRowStride, int
uvPixelStride, int[] out) {
        int i = 0;
        for (int y = 0; y < height; y++) {
            int pY = yRowStride * y;
            int uv_row_start = uvRowStride * (y >> 1);
            int pU = uv_row_start;
            int pV = uv_row_start;

            for (int x = 0; x < width; x++) {
                int uv_offset = (x >> 1) * uvPixelStride;
                out[i++] = YUV2RGB(
                        convertByteToInt(yData, pY + x),
                        convertByteToInt(uData, pU + uv_offset),
                        convertByteToInt(vData, pV + uv_offset));
            }
        }
    }

    private static int convertByteToInt(byte[] arr, int pos) {
        return arr[pos] & 0xFF;
    }

    private static int YUV2RGB(int nY, int nU, int nV) {
        nY -= 16;
        nU -= 128;
        nV -= 128;
```

```
        if (nY < 0) nY = 0;

        // This is the floating point equivalent. We do the conversion in integer
        // because some Android devices do not have floating point in hardware.
        // nR = (int)(1.164 * nY + 2.018 * nU);
        // nG = (int)(1.164 * nY - 0.813 * nV - 0.391 * nU);
        // nB = (int)(1.164 * nY + 1.596 * nV);

        int nR = (int) (1192 * nY + 1634 * nV);
        int nG = (int) (1192 * nY - 833 * nV - 400 * nU);
        int nB = (int) (1192 * nY + 2066 * nU);

        nR = Math.min(kMaxChannelValue, Math.max(0, nR));
        nG = Math.min(kMaxChannelValue, Math.max(0, nG));
        nB = Math.min(kMaxChannelValue, Math.max(0, nB));

        nR = (nR >> 10) & 0xff;
        nG = (nG >> 10) & 0xff;
        nB = (nB >> 10) & 0xff;

        return 0xff000000 | (nR << 16) | (nG << 8) | nB;
    }

    private static void fillBytes(final Image.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 || yuvBytes[i].length != buffer.capacity()) {
                yuvBytes[i] = new byte[buffer.capacity()];
            }
            buffer.get(yuvBytes[i]);
        }
    }


    public static void cropAndRescaleBitmap(final Bitmap src, final Bitmap dst, int
sensorOrientation) {
        Assert.assertEquals(dst.getWidth(), dst.getHeight());
        final float minDim = Math.min(src.getWidth(), src.getHeight());

        final Matrix matrix = new Matrix();
```

```
    // We only want the center square out of the original rectangle.
    final float translateX = -Math.max(0, (src.getWidth() - minDim) / 2);
    final float translateY = -Math.max(0, (src.getHeight() - minDim) / 2);
    matrix.preTranslate(translateX, translateY);

    final float scaleFactor = dst.getHeight() / minDim;
    matrix.postScale(scaleFactor, scaleFactor);

    // Rotate around the center if necessary.
    if (sensorOrientation != 0) {
        matrix.postTranslate(-dst.getWidth() / 2.0f, -dst.getHeight() / 2.0f);
        matrix.postRotate(sensorOrientation);
        matrix.postTranslate(dst.getWidth() / 2.0f, dst.getHeight() / 2.0f);
    }

    final Canvas canvas = new Canvas(dst);
    canvas.drawBitmap(src, matrix, null);
}

public static Matrix getTransformationMatrix(
        final int srcWidth,
        final int srcHeight,
        final int dstWidth,
        final int dstHeight,
        final int applyRotation,
        final boolean maintainAspectRatio) {
    final Matrix matrix = new Matrix();

    if (applyRotation != 0) {
        // Translate so center of image is at origin.
        matrix.postTranslate(-srcWidth / 2.0f, -srcHeight / 2.0f);

        // Rotate around origin.
        matrix.postRotate(applyRotation);
    }

    // Account for the already applied rotation, if any, and then determine how
    // much scaling is needed for each axis.
    final boolean transpose = (Math.abs(applyRotation) + 90) % 180 == 0;

    final int inWidth = transpose ? srcHeight : srcWidth;
```

```
        final int inHeight = transpose ? srcWidth : srcHeight;

        // Apply scaling if necessary.
        if (inWidth != dstWidth || inHeight != dstHeight) {
            final float scaleFactorX = dstWidth / (float) inWidth;
            final float scaleFactorY = dstHeight / (float) inHeight;

            if (maintainAspectRatio) {
                // Scale by minimum factor so that dst is filled completely while
                // maintaining the aspect ratio. Some image may fall off the edge.
                final float scaleFactor = Math.max(scaleFactorX, scaleFactorY);
                matrix.postScale(scaleFactor, scaleFactor);
            } else {
                // Scale exactly to fill dst from src.
                matrix.postScale(scaleFactorX, scaleFactorY);
            }
        }

        if (applyRotation != 0) {
            // Translate back from origin centered reference to destination frame.
            matrix.postTranslate(dstWidth / 2.0f, dstHeight / 2.0f);
        }

        return matrix;
    }
}
```

Logger.java

```
package org.tensorflow.demo.env;

import android.util.Log;

import java.util.HashSet;
import java.util.Set;

/**
 * Wrapper for the platform log function, allows convenient message prefixing and log disabling.
 */
public final class Logger {
  private static final String DEFAULT_TAG = "tensorflow";
  private static final int DEFAULT_MIN_LOG_LEVEL = Log.DEBUG;
```

```
  // Classes to be ignored when examining the stack trace
  private static final Set<String> IGNORED_CLASS_NAMES;

  static {
    IGNORED_CLASS_NAMES = new HashSet<String>(3);
    IGNORED_CLASS_NAMES.add("dalvik.system.VMStack");
    IGNORED_CLASS_NAMES.add("java.lang.Thread");
    IGNORED_CLASS_NAMES.add(Logger.class.getCanonicalName());
  }

  private final String tag;
  private final String messagePrefix;
  private int minLogLevel = DEFAULT_MIN_LOG_LEVEL;

  /**
   * Creates a Logger using the class name as the message prefix.
   *
   * @param clazz the simple name of this class is used as the message prefix.
   */
  public Logger(final Class<?> clazz) {
    this(clazz.getSimpleName());
  }

  /**
   * Creates a Logger using the specified message prefix.
   *
   * @param messagePrefix is prepended to the text of every message.
   */
  public Logger(final String messagePrefix) {
    this(DEFAULT_TAG, messagePrefix);
  }

  /**
   * Creates a Logger with a custom tag and a custom message prefix. If the message prefix
   * is set to <pre>null</pre>, the caller's class name is used as the prefix.
   *
   * @param tag identifies the source of a log message.
   * @param messagePrefix prepended to every message if non-null. If null, the name of the
caller is
   *                      being used
   */
```

```
public Logger(final String tag, final String messagePrefix) {
  this.tag = tag;
  final String prefix = messagePrefix == null ? getCallerSimpleName() : messagePrefix;
  this.messagePrefix = (prefix.length() > 0) ? prefix + ": " : prefix;
}

/**
 * Creates a Logger using the caller's class name as the message prefix.
 */
public Logger() {
  this(DEFAULT_TAG, null);
}

/**
 * Creates a Logger using the caller's class name as the message prefix.
 */
public Logger(final int minLogLevel) {
  this(DEFAULT_TAG, null);
  this.minLogLevel = minLogLevel;
}

public void setMinLogLevel(final int minLogLevel) {
  this.minLogLevel = minLogLevel;
}

public boolean isLoggable(final int logLevel) {
  return logLevel >= minLogLevel || Log.isLoggable(tag, logLevel);
}

/**
 * Return caller's simple name.
 *
 * Android getStackTrace() returns an array that looks like this:
 *      stackTrace[0]: dalvik.system.VMStack
 *      stackTrace[1]: java.lang.Thread
 *      stackTrace[2]: com.google.android.apps.unveil.env.UnveilLogger
 *      stackTrace[3]: com.google.android.apps.unveil.BaseApplication
 *
 * This function returns the simple version of the first non-filtered name.
 *
 * @return caller's simple name
 */
```

```
private static String getCallerSimpleName() {
  // Get the current callstack so we can pull the class of the caller off of it.
  final StackTraceElement[] stackTrace = Thread.currentThread().getStackTrace();

  for (final StackTraceElement elem : stackTrace) {
    final String className = elem.getClassName();
    if (!IGNORED_CLASS_NAMES.contains(className)) {
      // We're only interested in the simple name of the class, not the complete package.
      final String[] classParts = className.split("\\.");
      return classParts[classParts.length - 1];
    }
  }

  return Logger.class.getSimpleName();
}

private String toMessage(final String format, final Object... args) {
  return messagePrefix + (args.length > 0 ? String.format(format, args) : format);
}

public void v(final String format, final Object... args) {
  if (isLoggable(Log.VERBOSE)) {
    Log.v(tag, toMessage(format, args));
  }
}

public void v(final Throwable t, final String format, final Object... args) {
  if (isLoggable(Log.VERBOSE)) {
    Log.v(tag, toMessage(format, args), t);
  }
}

public void d(final String format, final Object... args) {
  if (isLoggable(Log.DEBUG)) {
    Log.d(tag, toMessage(format, args));
  }
}

public void d(final Throwable t, final String format, final Object... args) {
  if (isLoggable(Log.DEBUG)) {
    Log.d(tag, toMessage(format, args), t);
  }
```

```
  }

  public void i(final String format, final Object... args) {
    if (isLoggable(Log.INFO)) {
      Log.i(tag, toMessage(format, args));
    }
  }

  public void i(final Throwable t, final String format, final Object... args) {
    if (isLoggable(Log.INFO)) {
      Log.i(tag, toMessage(format, args), t);
    }
  }

  public void w(final String format, final Object... args) {
    if (isLoggable(Log.WARN)) {
      Log.w(tag, toMessage(format, args));
    }
  }

  public void w(final Throwable t, final String format, final Object... args) {
    if (isLoggable(Log.WARN)) {
      Log.w(tag, toMessage(format, args),t);
    }
  }

  public void e(final String format, final Object... args) {
    if (isLoggable(Log.ERROR)) {
      Log.e(tag, toMessage(format, args));
    }
  }

  public void e(final Throwable t, final String format, final Object... args) {
    if (isLoggable(Log.ERROR)) {
      Log.e(tag, toMessage(format, args), t);
    }
  }
}
```

MyUtility.java

```
package ai.fooz.foodanalysis.env;

import android.content.Context;
```

```
import android.content.DialogInterface;
import android.os.Bundle;
import android.support.v7.app.AlertDialog;
import android.view.View;

import com.google.firebase.analytics.FirebaseAnalytics;

import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.text.SimpleDateFormat;
import java.util.Date;
import java.util.Random;

import ai.fooz.foodanalysis.MainActivity;

public class MyUtility {

    private static final String LABEL_FILE = "file:///android_asset/labels.txt";
    public static final float PREDICTION_THRESHOLD = (float) 0.60;

    public static String getDateWithFormat(Date dt, String format) {
        SimpleDateFormat df = new SimpleDateFormat(format);
        String formattedDate = df.format(dt);
        return formattedDate;
    }

    public static String toTitleCase(String str) {

        if (str == null) {
            return null;
        }

        boolean space = true;
        StringBuilder builder = new StringBuilder(str);
        final int len = builder.length();

        for (int i = 0; i < len; ++i) {
            char c = builder.charAt(i);
            if (space) {
                if (!Character.isWhitespace(c)) {
                    // Convert to title case and switch out of whitespace mode.
```

```
                    builder.setCharAt(i, Character.toTitleCase(c));
                    space = false;
                }
            } else if (Character.isWhitespace(c)) {
                space = true;
            } else {
                builder.setCharAt(i, Character.toLowerCase(c));
            }
        }

        return builder.toString();
    }

    public static int getRandomNumber(int min, int max) {
        Random r = new Random();
        final int random = r.nextInt((max - min) + 1) + min;
        return random;
    }

    public static String getLabels(Context myContext) {

        String actualFilename = LABEL_FILE.split("file:///android_asset/")[1];
        String lbls = "";
        BufferedReader br = null;
        try {
            br = new BufferedReader(new
InputStreamReader(myContext.getAssets().open(actualFilename)));
            String line;
            while ((line = br.readLine()) != null) {
                lbls = lbls+"\n"+ MyUtility.toTitleCase(line);
            }
            br.close();
            return lbls;
        } catch (IOException e) {
            throw new RuntimeException("Problem reading label file!" , e);
        }
    }

    public static Boolean logFirebaseEvent(FirebaseAnalytics mFirebaseAnalytics, String
itemName) {
        Bundle bundle = new Bundle();
//          bundle.putString(FirebaseAnalytics.Param.ITEM_ID, id);
```

```
        bundle.putString(FirebaseAnalytics.Param.ITEM_NAME, itemName);
//          bundle.putString(FirebaseAnalytics.Param.CONTENT_TYPE, "image");
        mFirebaseAnalytics.logEvent(FirebaseAnalytics.Event.SELECT_CONTENT, bundle);
        return true;
    }
}
```

RecyclerTouch Listener.java

```
package ai.fooz.foodanalysis.env;

import android.content.Context;
import android.support.v7.widget.RecyclerView;
import android.view.GestureDetector;
import android.view.MotionEvent;
import android.view.View;

public class RecyclerTouchListener implements RecyclerView.OnItemTouchListener {

private GestureDetector gestureDetector;
private ClickListener clickListener;

public RecyclerTouchListener(Context context, final RecyclerView recyclerView, final
ClickListener clickListener) {
        this.clickListener = clickListener;
        gestureDetector = new GestureDetector(context, new
GestureDetector.SimpleOnGestureListener() {
@Override
public boolean onSingleTapUp(MotionEvent e) {
        return true;
        }

@Override
public void onLongPress(MotionEvent e) {
        View child = recyclerView.findChildViewUnder(e.getX(), e.getY());
        if (child != null && clickListener != null) {
        clickListener.onLongClick(child, recyclerView.getChildPosition(child));
        }
        }
        });
        }

@Override
public boolean onInterceptTouchEvent(RecyclerView rv, MotionEvent e) {
```

```
        View child = rv.findChildViewUnder(e.getX(), e.getY());
        if (child != null && clickListener != null && gestureDetector.onTouchEvent(e)) {
        clickListener.onClick(child, rv.getChildPosition(child));
        }
        return false;
        }

@Override
public void onTouchEvent(RecyclerView rv, MotionEvent e) {
        }

@Override
public void onRequestDisallowInterceptTouchEvent(boolean disallowIntercept) {

        }

public interface ClickListener {
    void onClick(View view, int position);

    void onLongClick(View view, int position);
}
}
```

Size.java

```
package org.tensorflow.demo.env;

import android.graphics.Bitmap;
import android.text.TextUtils;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

/**
 * Size class independent of a Camera object.
 */
public class Size implements Comparable<Size>, Serializable {

  // 1.4 went out with this UID so we'll need to maintain it to preserve pending queries when
  // upgrading.
```

```
public static final long serialVersionUID = 7689808733290872361L;

public final int width;
public final int height;

public Size(final int width, final int height) {
  this.width = width;
  this.height = height;
}

public Size(final Bitmap bmp) {
  this.width = bmp.getWidth();
  this.height = bmp.getHeight();
}

/**
 * Rotate a size by the given number of degrees.
 * @param size Size to rotate.
 * @param rotation Degrees {0, 90, 180, 270} to rotate the size.
 * @return Rotated size.
 */
public static Size getRotatedSize(final Size size, final int rotation) {
  if (rotation % 180 != 0) {
    // The phone is portrait, therefore the camera is sideways and frame should be rotated.
    return new Size(size.height, size.width);
  }
  return size;
}

public static Size parseFromString(String sizeString) {
  if (TextUtils.isEmpty(sizeString)) {
    return null;
  }

  sizeString = sizeString.trim();

  // The expected format is "<width>x<height>".
  final String[] components = sizeString.split("x");
  if (components.length == 2) {
    try {
      final int width = Integer.parseInt(components[0]);
      final int height = Integer.parseInt(components[1]);
```

```
            return new Size(width, height);
        } catch (final NumberFormatException e) {
            return null;
        }
    } else {
        return null;
    }
}

public static List<Size> sizeStringToList(final String sizes) {
    final List<Size> sizeList = new ArrayList<Size>();
    if (sizes != null) {
        final String[] pairs = sizes.split(",");
        for (final String pair : pairs) {
            final Size size = Size.parseFromString(pair);
            if (size != null) {
                sizeList.add(size);
            }
        }
    }
    return sizeList;
}

public static String sizeListToString(final List<Size> sizes) {
    String sizesString = "";
    if (sizes != null && sizes.size() > 0) {
        sizesString = sizes.get(0).toString();
        for (int i = 1; i < sizes.size(); i++) {
            sizesString += "," + sizes.get(i).toString();
        }
    }
    return sizesString;
}

public final float aspectRatio() {
    return (float) width / (float) height;
}

@Override
public int compareTo(final Size other) {
    return width * height - other.width * other.height;
}
```

```
  @Override
  public boolean equals(final Object other) {
    if (other == null) {
      return false;
    }

    if (!(other instanceof Size)) {
      return false;
    }

    final Size otherSize = (Size) other;
    return (width == otherSize.width && height == otherSize.height);
  }

  @Override
  public int hashCode() {
    return width * 32713 + height;
  }

  @Override
  public String toString() {
    return dimensionsAsString(width, height);
  }

  public static final String dimensionsAsString(final int width, final int height) {
    return width + "x" + height;
  }
}
```

SplitTimer.java

```
package org.tensorflow.demo.env;

import android.os.SystemClock;

/**
 * A simple utility timer for measuring CPU time and wall-clock splits.
 */
public class SplitTimer {
  private final Logger logger;
```

```
  private long lastWallTime;
  private long lastCpuTime;

  public SplitTimer(final String name) {
    logger = new Logger(name);
    newSplit();
  }

  public void newSplit() {
    lastWallTime = SystemClock.uptimeMillis();
    lastCpuTime = SystemClock.currentThreadTimeMillis();
  }

  public void endSplit(final String splitName) {
    final long currWallTime = SystemClock.uptimeMillis();
    final long currCpuTime = SystemClock.currentThreadTimeMillis();

    logger.i(
        "%s: cpu=%dms wall=%dms",
        splitName, currCpuTime - lastCpuTime, currWallTime - lastWallTime);

    lastWallTime = currWallTime;
    lastCpuTime = currCpuTime;
  }
}
```

TensorFlowImageClassifier.java

```
package ai.fooz.foodanalysis.env;

import android.content.res.AssetManager;
import android.graphics.Bitmap;
import android.os.Trace;
import android.util.Log;
import java.io.BufferedReader;
import java.io.IOException;
import java.io.InputStreamReader;
import java.util.ArrayList;
import java.util.Comparator;
import java.util.List;
import java.util.PriorityQueue;
```

```
import java.util.Vector;
import org.tensorflow.Operation;
import org.tensorflow.contrib.android.TensorFlowInferenceInterface;

/** A classifier specialized to label images using TensorFlow. */
public class TensorFlowImageClassifier implements Classifier {
  public static final String TAG = "TensorFlowImageClassifier";

  // Only return this many results with at least this confidence.
  public static final int MAX_RESULTS = 3;
  public static final float THRESHOLD = 0.1f;

  // Config values.
  public String inputName;
  public String outputName;
  public int inputSize;
  public int imageMean;
  public float imageStd;

  // Pre-allocated buffers.
  public Vector<String> labels = new Vector<String>();
  public int[] intValues;
  public float[] floatValues;
  public float[] outputs;
  public String[] outputNames;

  public boolean logStats = false;

  public TensorFlowInferenceInterface inferenceInterface;

  public TensorFlowImageClassifier() {}

  /**
   * Initializes a native TensorFlow session for classifying images.
   *
   * @param assetManager The asset manager to be used to load assets.
   * @param modelFilename The filepath of the model GraphDef protocol buffer.
   * @param labelFilename The filepath of label file for classes.
   * @param inputSize The input size. A square image of inputSize x inputSize is assumed.
   * @param imageMean The assumed mean of the image values.
   * @param imageStd The assumed std of the image values.
   * @param inputName The label of the image input node.
```

```
 * @param outputName The label of the output node.
 * @throws IOException
 */
public static Classifier create(
    AssetManager assetManager,
    String modelFilename,
    String labelFilename,
    int inputSize,
    int imageMean,
    float imageStd,
    String inputName,
    String outputName) {
  TensorFlowImageClassifier c = new TensorFlowImageClassifier();
  c.inputName = inputName;
  c.outputName = outputName;

  // Read the label names into memory.
  // TODO(andrewharp): make this handle non-assets.
  String actualFilename = labelFilename.split("file:///android_asset/")[1];
  Log.i(TAG, "Reading labels from: " + actualFilename);
  BufferedReader br = null;
  try {
    br = new BufferedReader(new InputStreamReader(assetManager.open(actualFilename)));
    String line;
    while ((line = br.readLine()) != null) {
      c.labels.add(line);
    }
    br.close();
  } catch (IOException e) {
    throw new RuntimeException("Problem reading label file!" , e);
  }

  c.inferenceInterface = new TensorFlowInferenceInterface(assetManager, modelFilename);

  // The shape of the output is [N, NUM_CLASSES], where N is the batch size.
  final Operation operation = c.inferenceInterface.graphOperation(outputName);
  final int numClasses = (int) operation.output(0).shape().size(1);
  Log.i(TAG, "Read " + c.labels.size() + " labels, output layer size is " + numClasses);

  // Ideally, inputSize could have been retrieved from the shape of the input operation. Alas,
  // the placeholder node for input in the graphdef typically used does not specify a shape, so it
  // must be passed in as a parameter.
```

```
    c.inputSize = inputSize;
    c.imageMean = imageMean;
    c.imageStd = imageStd;

    // Pre-allocate buffers.
    c.outputNames = new String[] {outputName};
    c.intValues = new int[inputSize * inputSize];
    c.floatValues = new float[inputSize * inputSize * 3];
    c.outputs = new float[numClasses];

    return c;
  }

  @Override
  public List<Recognition> recognizeImage(final Bitmap bitmap) {
    // Log this method so that it can be analyzed with systrace.
    Trace.beginSection("recognizeImage");

    Trace.beginSection("preprocessBitmap");
    // Preprocess the image data from 0-255 int to normalized float based
    // on the provided parameters.
    bitmap.getPixels(intValues, 0, bitmap.getWidth(), 0, 0, bitmap.getWidth(),
bitmap.getHeight());
    for (int i = 0; i < intValues.length; ++i) {
      final int val = intValues[i];
      floatValues[i * 3 + 0] = (((val >> 16) & 0xFF) - imageMean) / imageStd;
      floatValues[i * 3 + 1] = (((val >> 8) & 0xFF) - imageMean) / imageStd;
      floatValues[i * 3 + 2] = ((val & 0xFF) - imageMean) / imageStd;
    }
    Trace.endSection();

    // Copy the input data into TensorFlow.
    Trace.beginSection("feed");
    inferenceInterface.feed(inputName, floatValues, 1, inputSize, inputSize, 3);
    Trace.endSection();

    // Run the inference call.
    Trace.beginSection("run");
    inferenceInterface.run(outputNames, logStats);
    Trace.endSection();

    // Copy the output Tensor back into the output array.
```

```
  Trace.beginSection("fetch");
  inferenceInterface.fetch(outputName, outputs);
  Trace.endSection();

  // Find the best classifications.
  PriorityQueue<Recognition> pq =
      new PriorityQueue<Recognition>(
          3,
          new Comparator<Recognition>() {
            @Override
            public int compare(Recognition lhs, Recognition rhs) {
              // Intentionally reversed to put high confidence at the head of the queue.
              return Float.compare(rhs.getConfidence(), lhs.getConfidence());
            }
          });
  for (int i = 0; i < outputs.length; ++i) {
    if (outputs[i] > THRESHOLD) {
      pq.add(
          new Recognition(
              "" + i, labels.size() > i ? labels.get(i) : "unknown", outputs[i], null));
    }
  }
  final ArrayList<Recognition> recognitions = new ArrayList<Recognition>();
  int recognitionsSize = Math.min(pq.size(), MAX_RESULTS);
  for (int i = 0; i < recognitionsSize; ++i) {
    recognitions.add(pq.poll());
  }
  Trace.endSection(); // "recognizeImage"
  return recognitions;
}

@Override
public void enableStatLogging(boolean logStats) {
  this.logStats = logStats;
}

@Override
public String getStatString() {
  return inferenceInterface.getStatString();
}

@Override
```

```
  public void close() {
    inferenceInterface.close();
  }
}
```

```
package ai.fooz.models;

import com.orm.SugarRecord;

import java.util.Date;

public class Prediction extends SugarRecord {
    public String title;
    public String confidence;
    public Integer calories;
    public Integer carbs;
    public Integer fats;
    public Integer proteins;
    public Boolean isSelected;
    public RefImage refimage;

    public Prediction() {

    }
}
```

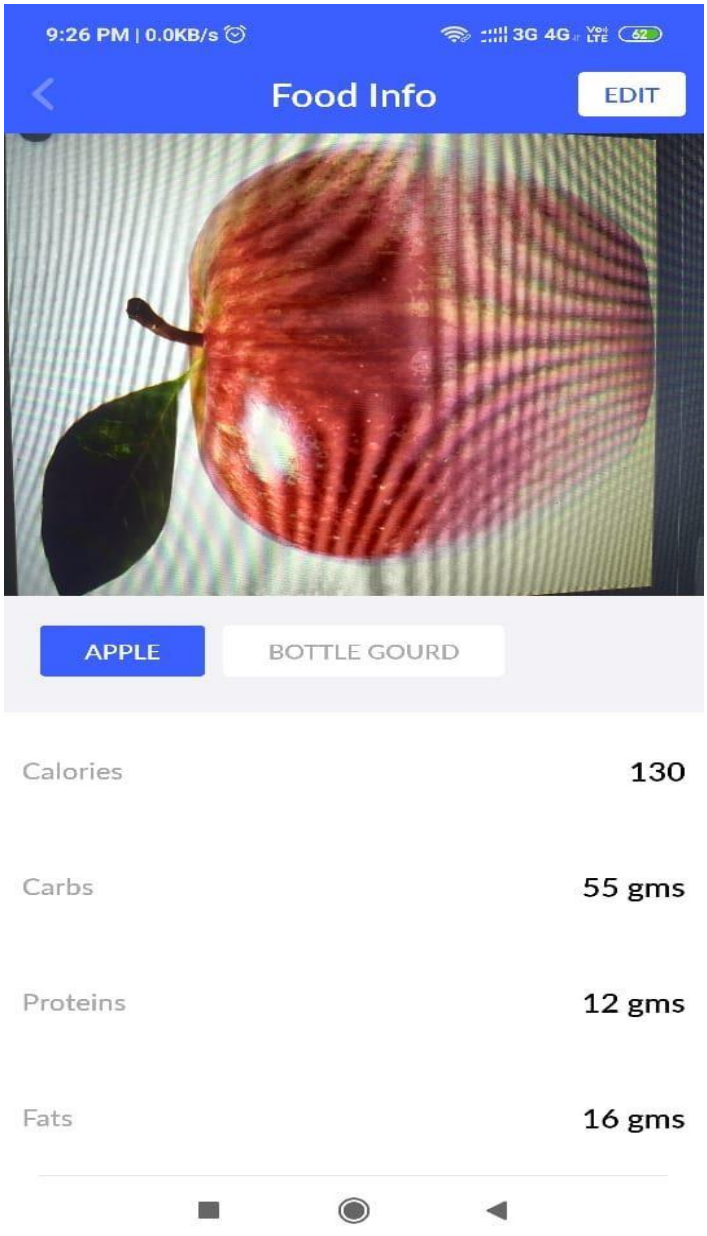
RefImage.java

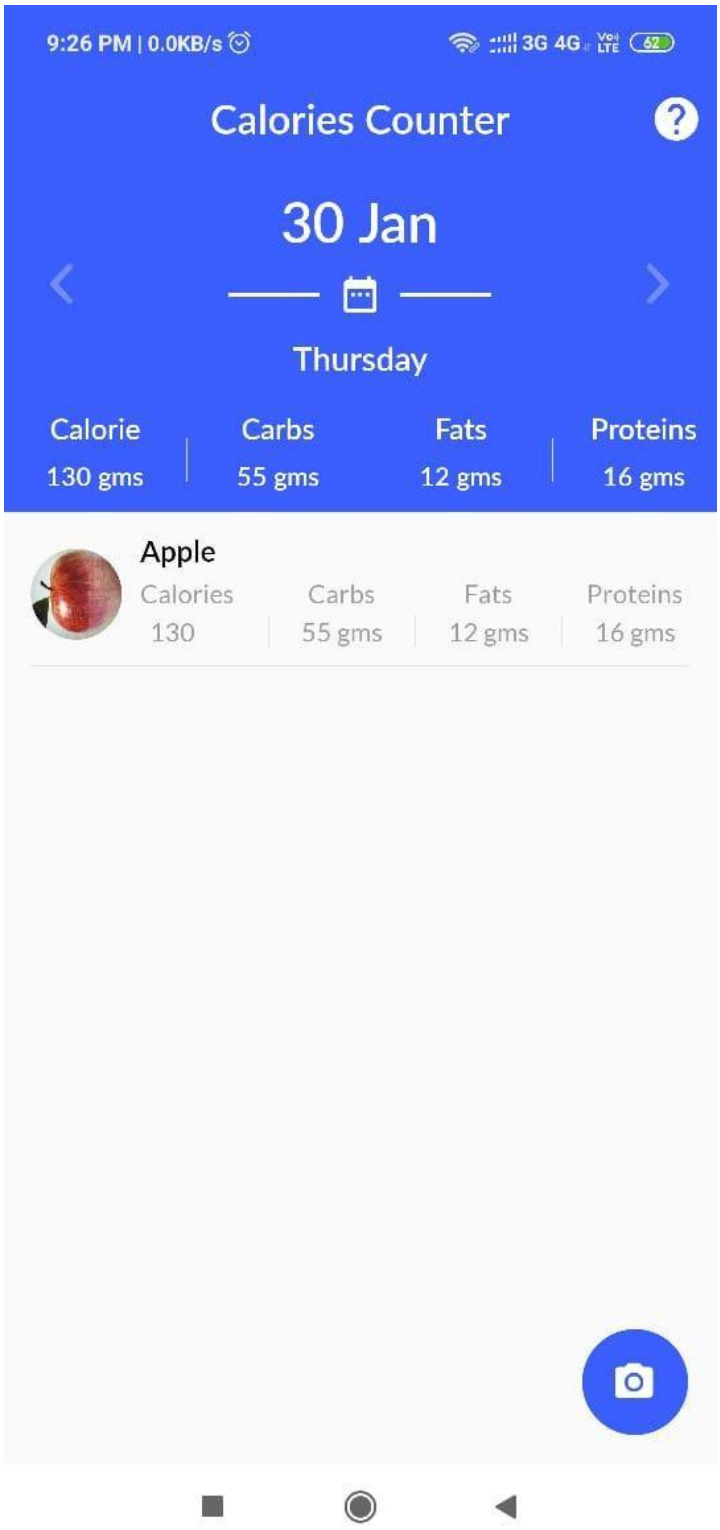
```
package ai.fooz.models;

import android.media.Image;

import com.orm.SugarRecord;

import java.util.Date;
import java.util.List;

public class RefImage extends SugarRecord {
    public String name;
    public String timestamp;

    public RefImage() {

    }

    public RefImage(String name) {
        this.name = name;
    }
```

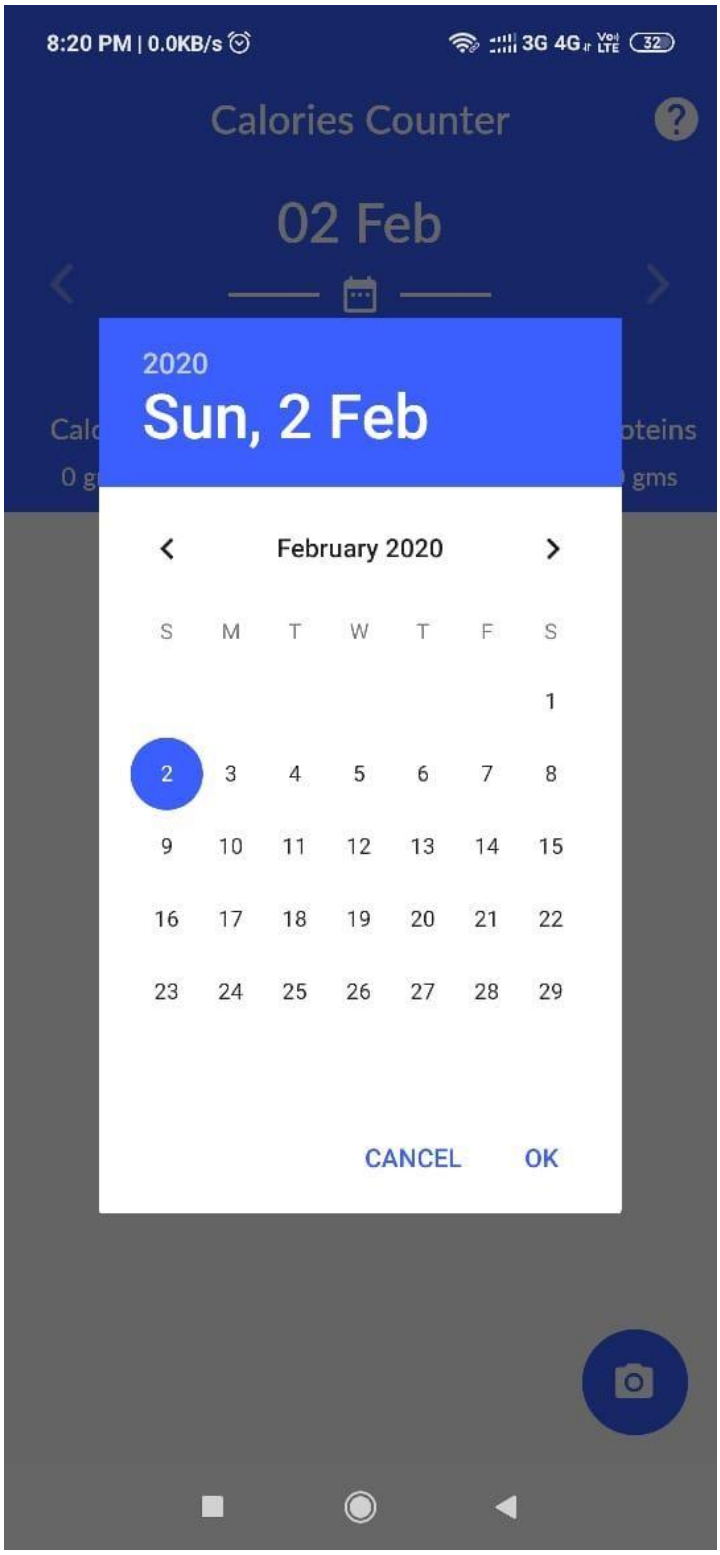
```
    public List<Prediction> getPredictions(){
        return Prediction.find(Prediction.class, "refimage = ?", getId().toString());
    }

    public int getSelectedPredPosition(){
        int spos = 0;
        List<Prediction> ls = getPredictions();
        for(int  i=0;i<ls.size();i++){
            Prediction pd = ls.get(i);
            if(pd.isSelected){
                spos = i;
                break;
            }
        }
        return spos;
    }
//
//     public void setPredictions(List<Prediction> predictions) {
//
//         return Prediction.find(Prediction.class, "refimage = ?", getId().toString());
//     }
}
```

APPENDIX B

SCREENSHOTS:




9:26 PM | 0.0KB/s
3G 4G
Calories Counter
30 Jan
Thursday
Calorie
130 gms
Carbs
55 gms
Fats
12 gms
Proteins
16 gms
Apple
Calories
130
Carbs
55 gms
Fats
12 gms
Proteins
16 gms


8:20 PM | 0.0KB/s
3G 4G
Calories Counter
02 Feb
2020
Sun, 2 Feb
February 2020
S M T W T F S
1
2 3 4 5 6 7 8
9 10 11 12 13 14 15
16 17 18 19 20 21 22
23 24 25 26 27 28 29
CANCEL
OK

X. REFERENCES

[1] Abadi, M., Agarwal, A., Barham, P., Brevdo, E., & Zheng, X. (2016).Tensor Flow: Large-scale machine learning on heterogeneous distributed systems. Retrieved from http://arxiv.org/abs/1603.04467 Ahmed,A.,&Ozeki,T.(2015).

[2] Food image recognition by using Bag-of-SURF features and HOG Features. In Proceedings of the 3rd International Conference on Human-Agent Interaction (pp. 179–

[3]https://doi.org/10.1145/2814940.2814968 Al-Sarayreh, M., Reis, M. M., Yan, W. Q., & Klette, R. (2018).

[4] Detection of red-meat adulteration by deep spectral-spatial features in hyperspectral images. Journal of Imaging,

[5] https://doi.org/10.3390/jimaging4050063 Azizah, L. M., Umayah, S. F., Riyadi, S., Damarjati, C., & Utama, N. A. (2017).

[6]W. Wu and J. Yang, "Fast food recognition from videos of eating for calorie estimation," in Multimedia and Expo, 2009. ICME 2009. IEEE International Conference on. IEEE, 2009, pp. 1210–1213.

[7] N. Yao, R. J. Sclabassi, Q. Liu, J. Yang, J. D. Fernstrom, M. H. Fernstrom, and M. Sun, "A video processing approach to the study of obesity," in Multimedia and Expo, 2007 IEEE International Conference on. IEEE, 2007, pp. 1727–1730.

[8] S. Yang, M. Chen, D. Pomerleau, and R. Sukthankar, "Food recognition using statistics of pairwise local features," in Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on. IEEE, 2010, pp. 2249–2256.

[9] M. Bosch, F. Zhu, N. Khanna, C. J. Boushey, and E. J. Delp, "Combining global and local features for food identification in dietary assessment," in Image Processing (ICIP), 2011 18th IEEE International Conference on. IEEE, 2011, pp. 1789–1792.