Motu-Petu: A Food Recognition App

Mobile Application Engineering, Fall 2018

Final Project Report

Ishank Sharma (is356)
Shubham Jain(sj671)
Ashwin Channakeshava(ak1645)
Department of Electrical and Computer Engineering
Rutgers University, New Brunswick

1 Introduction

We introduce an app that is designed especially for food enthusiasts and travellers. Our app allows users to take picture of the targeted dish and then recognizes the dish from our database of hundreds of different food items. The app also allows to share the captured Image to Facebook, as a traveller might like to share his food experience with friends. We explored Camera2 as a novel API and also integrated an exterior application call through Facebook's API. Refer Figure 1 for screenshots of our app.

2 Background

2.1 Image Classification

To begin, our problem aims to recognize and classify food images from various cuisines. The system should assign (potentially multiple) semantic labels to an image in an unconstrained setting. To learn about thousands of objects from millions of images, we need a model with a significant learning capacity. To tackle this, we've chosen a class of Deep Neural Networks known as Convolutional Neural Networks (CNN) [1] [2]. Their capacity can be controlled by varying their depth and breadth, and they also make mostly correct assumptions about the nature of images (namely,

Figure 1: Screenshot of Motu-Petu Food Recognition App

Type / Stride	Filter Shape	Input Size
Conv / s2	$3 \times 3 \times 3 \times 32$	$224 \times 224 \times 3$
Conv dw / s1	$3 \times 3 \times 32$ dw	$112 \times 112 \times 32$
Conv / s1	$1 \times 1 \times 32 \times 64$	$112 \times 112 \times 32$
Conv dw / s2	$3 \times 3 \times 64$ dw	$112 \times 112 \times 64$
Conv / s1	$1 \times 1 \times 64 \times 128$	$56 \times 56 \times 64$
Conv dw / s1	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 128$	$56 \times 56 \times 128$
Conv dw / s2	$3 \times 3 \times 128$ dw	$56 \times 56 \times 128$
Conv / s1	$1 \times 1 \times 128 \times 256$	$28 \times 28 \times 128$
Conv dw / s1	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 256$	$28 \times 28 \times 256$
Conv dw / s2	$3 \times 3 \times 256$ dw	$28 \times 28 \times 256$
Conv / s1	$1 \times 1 \times 256 \times 512$	$14 \times 14 \times 256$
$5\times$ Conv dw / s1	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 512$	$14 \times 14 \times 512$
Conv dw / s2	$3 \times 3 \times 512$ dw	$14 \times 14 \times 512$
Conv / s1	$1 \times 1 \times 512 \times 1024$	$7 \times 7 \times 512$
Conv dw / s2	$3 \times 3 \times 1024$ dw	$7 \times 7 \times 1024$
Conv / s1	$1 \times 1 \times 1024 \times 1024$	$7 \times 7 \times 1024$
Avg Pool / s1	Pool 7×7	$7 \times 7 \times 1024$
FC / s1	1024×1000	$1 \times 1 \times 1024$
Softmax / s1	Classifier	$1 \times 1 \times 1000$

Figure 2: MobileNet Architectue adpated from [5]

stationary of statistics and locality of pixel dependencies). In many real-world applications such as self-driving car, robotics, and augmented reality, the classification tasks need to be carried out in a timely fashion on a computationally limited platform. Considering the memory and time constraints, we can not use a generalized variant of CNN architectures like- InceptionV4 [3], ResNeXt, VGG16 [4] Google recently proposed a MobileNet [5], a family of mobile-first computer vision models for TensorFlow, designed to effectively maximise accuracy while being mindful of the restricted resources for an on-device or embedded application. MobileNets are small, low-latency, low-power models parameterized to address the resource restrictions of a variety of use cases. Refer Figure 2 for MobileNet Architecture.

We use MobileNet V1 version of CNN for food image classification and fast inference. It obtained top-1 accuracy of 70.9% and top-5 accuracy of 89.9% on ILSVRC-2012 dataset.

To our knowledge there is only one app- CalorieMama [6] in Google Play store which applies food classification and nutrition tracking for users.

2.2 Camera2 API

The Camera2 API models camera as a pipeline. It takes input requests to capture a frame or a image and provides captured result and meta-data as buffer. There are multiple classes under Camera2 API that are required for its usage. Most classes have callback functions which when used to define a new callback, provides us points of interaction with camera. We started with Surface Texture listener, this keeps status of textureView through course of activity. Next we setup the camera-device and it's state callback. We defined operations for camera when opened, closed. Since Image capturing and other camera operations need to be handled seperately apart from primary application flow, we need to use a handler and handler thread to address this. Once through the initial setup of camera, we now setup our device, this requires to configure camera characteristics. We point to the rear camera using camera manager and pass it to our connection request using Camera characteristics. Next we setup the preview orientations and basic image scaling and rotation. The camera output will be displayed through capture sessions, the final image is also a snapshot of this session that we get using the request builder. To fetch the captured image we use imagereader which provides on Image available listener to grab image from capture session. For file storage we use ImageUtils class, where we defined file name to be Appname_datestamp.jpg.

2.3 Photo Sharing

Facebook provides a very popular widget for applications to use to share media. In the application that we have developed, we decided not to use this widget as we wanted to provide more space for the camera interface. Instead, when the volume down button is pressed, the application automatically opens the share page with the image captured on Facebook. This allows the user to post the image with the name of the dish that he or she now knows courtesy of our application.

We have included Facebook integration [7] in our mobile application. To do this, the first step is to create a facebook app ID on developers.facebook.com. After obtaining the app ID we can integrate the facebook SDK in our application. To do this, we need to include the facebook application ID in the resources file that is strings.xml. The next step is to include the Facebook content provider with the facebook app ID in the manifest file. In the same file, we have to include permission for the application to be able to access the internet. We also have to include a meta data element in the application element in the manifest file. We have to include the implementation of the facebook SDK in the gradle build file.

The next step is to create a data key hash from the mobile application. A function is included in the main Activity to generate a hash key from the application. This hash key can be viewed from the logcat in android studio. This hash key has to be provided to facebook on facebook.developers.com for the application to be able to access facebook. Once this process is done, the mobile application will have access to share images on facebook.

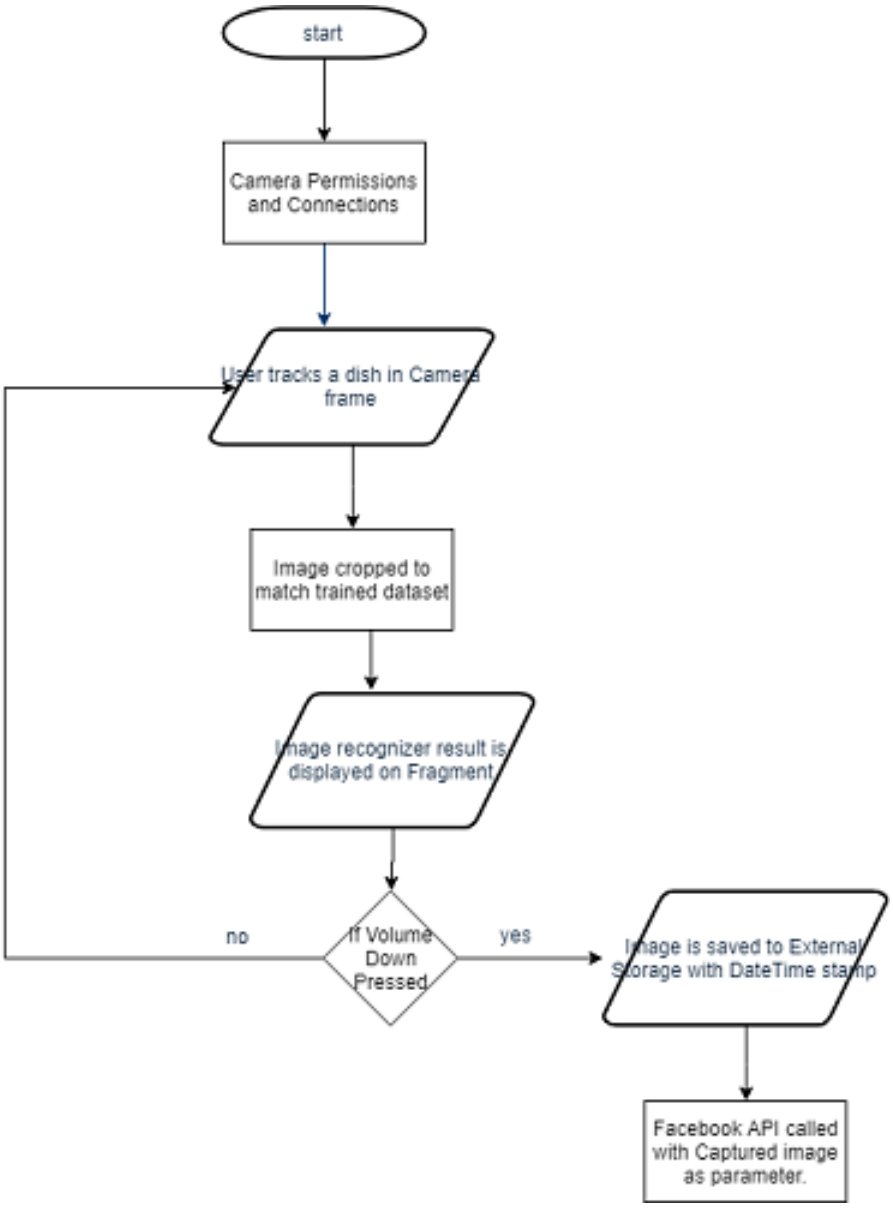

Figure 3: MotuPetu Application Pipeline

3 Application Pipeline

An overview of our application pipeline is shown in Figure 3. Our application is divided among various classes that handles various image sensing, image recognizing and image handling operations. Classes such as BorderedText, ImageUltils, Logger, Size, SplitTimer are basic functionality and utility classes that we have extended for our use. We use these to display image result text and storage operations. The class AutoFitTextureView is used for making application compatible with more phones with respect to layout. The CameraActivity class seeks permission for camera, uses preview image . The Classifier class holds declarations for our classifier. The ClassifierAcitvity is

our home activity, this is the class where the image is passed to classifier, fetches resullt and call storage methods if Volume key down is pressed. Class RecognitionScoreView sets the view with the result and TensorFlowImageClassifer is the class that is doing the recognition comparison with our trained model.

4 Model Training

4.1 Datasets

Deep learning bases algorithms require a large dataset for training efficiently. For our task, we have used two publically available datasets, namely- UECFOOD256 [8] and FOOD101 [9]. A summary of these datasets is as follows-

- UECFOOD256: The dataset contains 256-kind food photos from Japanese and Korean cuisines. There are 150 images per food category. I addition, each food photo has a bounding box indicating the location of the food item in the photo.
- FOOD101: Thes dataset has 101 food categories, with 101,000 images. Each dish class has 1000 sample images.

Combining these two datasets, we have 139400 food dish images and 357 food categories. The dataset was further divided into 60%-20%-20% train/dev/test split. The dev set was used for validation and model refinement We scraped the labels for each category from provided text files. No image filtering of any kind was applied since we want to obtain a generalizable model which can perform effective classification with varying ambient conditions and image content.

4.2 Classification Model

Training a deep neural network from scratch often takes several weeks. A promising alternative to training a CNN from scratch is to fine-tune a CNN that has been trained using a massive labelled dataset from a different application. The pre-trained models have been applied successfully to various computer vision tasks as a feature generator or as a baseline for transfer learning [10]. We approach our problem by using a pretrained MobileNet model. The model is publically available and named- **mobilenet_1.0_224**. In our model, we finetune the last softmax classification layer of MobileNet and freeze the network weights for all other layers. The model was built with Tensorflow Mobile library [11] and Python 3.6 as a programming language.

During training, we randomly left flip or right flip images and cropped the size to 224x224 before feeding it to the input layer.

The network was trained on Nvidia GTX1060Ti for 7400 epochs with learning rate 0.01 and batch size 256. Rmsprop is used as choice of optimizer inline with Inception V3 network. We obtained moderate accuracies - Validation Top-1 44.1% and Test Top-1 43.7%. Refer Figure 4 for train accuracy and Figure 5 for validation accuracy profile. From our experiments, we found that even when the recognition score is quite low, the predicted class matches the food item. Therefore, we

set the classification confidence threshold to be greater than 0. Please refer Figure 1 for one such example.

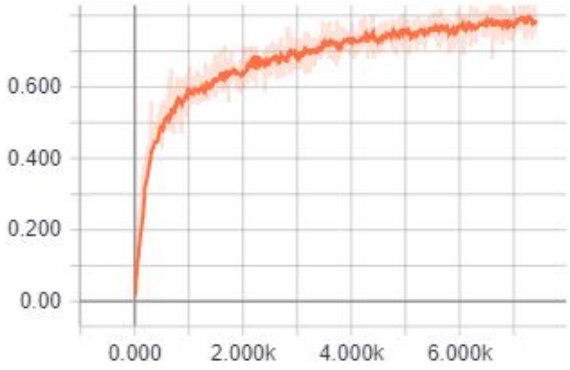

Figure 4: Train accuracy for 7400 epochs

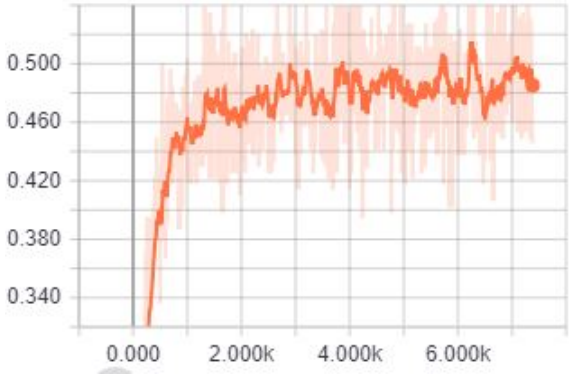

Figure 5: Val accuracy for 7400 epochs

We further faced challenges in deploying the model directly to mobile. There were many unneeded operations in the frozen graph (trained model). For interfacing with the Android development environment, the graph needs to be further trimmed to make it small. Tensorflow provides a script- *strip_unused.py*, which is available online for stripping unrequired nodes. In our code TensorFlowImageClassifier.java defines the classification utilities and Classifier.java is an interface class for the classification activity. According to our experiments, the deployed model recognises a food image within 140-152 ms.

5 Team-Work

We divided ourselves based upon our skills for each section of application. Ishank built scripts to train data model, while Shubham and Ashwin studied camera 2 API and facebook API. Not only the image capturing was a step of Ishank but we had to share our understanding about how some value will be treated in functions made by each of us. The primary challenges were mostly integration issues as whatever we have coded failed to run with each other at first run. We then

studied through various other code builds and methods to implement the same logic into different structure.

6 Future-Scope

- The application classifier has managed to deliver good results but we believe that this can be improved by using MobileNet V2. Moreover, the network memory and parameter size can be further reduced with techniques like 8 bit Quantization, Inverted bottlenecks.
- Camera features currently doesn't use many camera characterstics, features such as auto exposure, auto focus etc. that improves image quality can be applied. Also, for post production process of image filters can be added as well.
- The app works right now for only one facebook account, this can be extended to use multiple facebook accounts, as well as to integrate other popular social platforms.
- For our future work we want to explore connecting the classification result with a food nutrition or recipe database to track calories and nutrition. This way user can track nutrition and prepare new dishes. Furthermore, bad recognition results can be tagged by users which can be used as feedback for online learning of Deep Neural Network which will in turn increase recognition accuracy.

References

[1] K. Fukushima and S. Miyake, "Neocognitron: A self-organizing neural network model for a mechanism of visual pattern recognition," in *Competition and cooperation in neural nets*. Springer, 1982, pp. 267–285.

[2] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[3] C. Szegedy, S. Ioffe, V. Vanhoucke, and A. A. Alemi, "Inception-v4, inception-resnet and the impact of residual connections on learning." in *AAAI*, vol. 4, 2017, p. 12.

[4] J. Redmon and A. Farhadi, "Yolo9000: better, faster, stronger," *arXiv preprint*, 2017.

[5] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "Mobilenets: Efficient convolutional neural networks for mobile vision applications," *arXiv preprint arXiv:1704.04861*, 2017.

[6] "Caloriemama." [Online]. Available: https://www.caloriemama.ai/app

[7] "Facebookandroid sdk." [Online]. Available: https://developers.facebook.com/docs/android/getting-started#create_hash

[8] L. Bossard, M. Guillaumin, and L. Van Gool, "Food-101 – mining discriminative components with random forests," in *European Conference on Computer Vision*, 2014.

[9] Y. Kawano and K. Yanai, "Automatic expansion of a food image dataset leveraging existing categories with domain adaptation," in *Proc. of ECCV Workshop on Transferring and Adapting Source Knowledge in Computer Vision (TASK-CV)*, 2014.

[10] N. Wang, S. Li, A. Gupta, and D.-Y. Yeung, "Transferring rich feature hierarchies for robust visual tracking," *arXiv preprint arXiv:1501.04587*, 2015.

[11] "Tensorflow mobile." [Online]. Available: https://www.tensorflow.org/lite/