RiddleMe

Mitchell Youker

Advisor: Dr. Michael J. Reale

December 2017

Abstract

The purpose of this project is to develop a mobile game application where users can identify objects within their camera's viewpoint to receive points. Each point is awarded if the user can solve a "riddle" for that geolocation. The riddle is solved when the user places their camera in front of the specified object. Users can also create riddles, as well as solve riddles that other users have created. The focus of this paper will touch on how the mobile application will detect objects through a users camera interface.

1 Introduction

The largest challenge with this project is how to efficiently classify object via a camera interface with mobile hardware. Computer vision in general can be a very taxing on the hardware for even a desktop environment. Due to the limitation of mobile hardware, computer vision applications must be developed in a very optimized fashion to achieve effective results. Upon first examining this problem I researched Oriented FAST and Rotated BRIEF (ORB) [1]. ORB is very similar to SIFT and SURF in that they all extracts key points from an image, but ORB claims that it operates two orders of magnitude faster than SIFT, making for great potential candidate to solve the object detection problem. ORB however, requires extra work to turn it into a classifier as for it is natively only a key point extractor. ORB also only extracts key points from a single image and therefore will have a harder time extracting key points from different distances from the object as well as different orientations of the object. Due to these reasons, I decided to research neural network architectures that would provide better generalize of the objects.

2 Method

The first architecture I researched was Alexnet [2]. Alexnet was one of the first neural network architectures that I had come across. The paper states that Alexnet preforms exceptionally well when trained on ImageNet for classifying many different classes. The paper also states that Alexnet can be trained much faster than traditional neural networks by utilizing ReLu functions as opposed to Sigmoid or a Tanh functions which reduces the computations need when retraining the weights (reduces vanishing gradient problem). Alexnet is great for large classification, but not ideal to be run on a mobile application as it requires a lot of computations (GPU). Alexnet was modified by Google and named GoogleNet and later called the Inception model which can be found in the Android Tensorflow example application [3]. Although this model did perform results that are sufficient, it was utilizing almost 80% CPU usage on a Samsung Galaxy S8 making for very laggy results as well as poor battery quality. I found that Google released a new model called MobileNet that was created to address the issues related to CPU usage (leads to lower frame rate) to create real-time mobile applications [4]. MobileNet assumes that most of the weights in the model are rarely used (low activation function) and therefore removes them from the final model to create a faster model that is more compressed. I chose to go with MobileNet for these reasons and de-

cided to use the native architecture along with the retrain tool provided by the Tensorflow repository [5]. The retrain tool retrains the final layer of a model to your classes reducing training time significantly. This worked out perfect for testing different flavors of MobileNet architecture (different input image sizes and number of weights) as well as the learning rate, training steps and data augmentation. The retrain tool outputs a frozen model of your retrained network (protobufs; .pb file) which fits very nicely with Android Tensorflow example application as it uses a frozen model of the Inception V3 architecture trained on ImageNet [**?**, tensorflow] I chose to use the Tensorflow Android Classifier example application as the boilerplate for this project. The main view will show a Google map interface displaying riddles relative to the users location. Users can select markers to display their riddles (Seen in figure 3). When a user selects a marker 25 meters away from them a button will appear allowing them to open the camera interface and solve the riddle. The Android application also depict a navigation drawer that display the user's information (Name, pictures, etc.) and his or her number of solved riddles as well as buttons to create a new riddle or go back to the Google map interface. The riddles are stored into a PostgreSQL database using the PostGIS extension for easier and faster querying. Figure 1 displays how the PostgreSQL tables are structured via an ERD diagram. I used PHP to communicate information between the database and the Android application via HTTP/JSON responses. The workflow of the project can be seen in Figure 2.

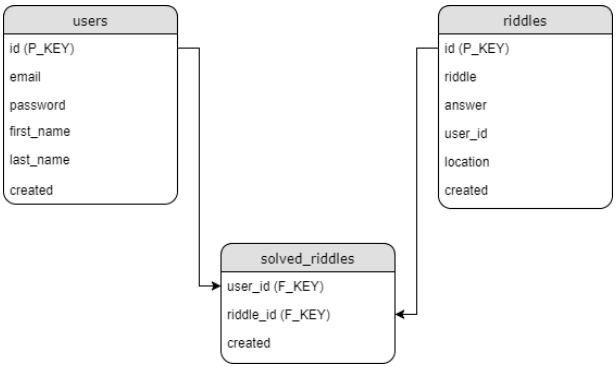


Figure 1: ERD of SQL tables.

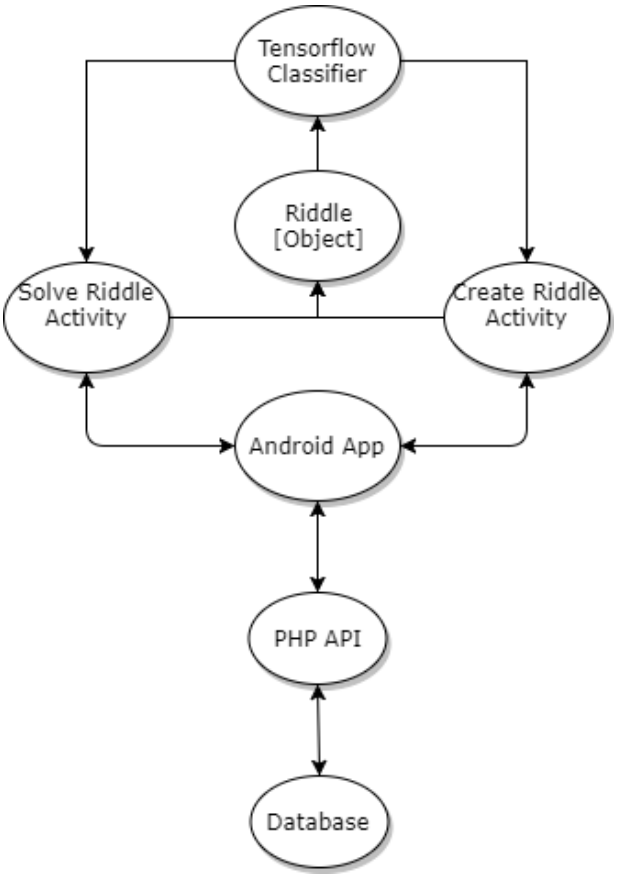


Figure 2: The workflow of the project.

3 Experiments

All experiments were done using a Nvidia GTX 970 with CUDA 8.0. The Android application was developed on a Samsung Galaxy S8. I chose to use the MobileNet 224 input size model as this produces the most accurate model, but at the cost of storage (20MB). I will be using ImageNet for my data set [7]. Each of the classes were downloaded individually from ImageNet for retraining purposes. The following classes were used basketball hoop, bridge, chair,

computer, desk, door, goal, grill, laundry machine, podium, projector, scoreboard, sign, stair, statue, street light, swing, trash can, TV, vending machine and water fountain. I split the experiment between four trials.

3.1 Trial 1

I chose a learning rate of 0.0001, 32 images per training batch, 1000 training steps and set the validation batch size to use the entire validation set (for more accurate results). I split the data set between 20% validation, 20% training and did not use data augmentation for my initial test.

3.2 Trial 2

I kept the same settings from trial 1 and adjusting the learning rate to 0.005 to better understand how the learning rate was affecting the model. I found overfitting occurs around 800-900 training steps.

3.3 Trial 3

For the third trial I reduced the training steps to 900 and added in data augmentation parameters for random scaling at a max of 10%, random brightness at a max of 10% and flipping of images horizontally. I found that the data augmentation created slightly better accuracy.

3.4 Trial 4

As an attempt to gain more accurate results, for the fourth trial I added in a miscellaneous class that is not relevant to the other classes. This miscellaneous class contains birds, books, dogs, fire matches and scorpions from ImageNet. If the miscellaneous class is detected via the Android classifier, the result would be dropped as we are only interested in the classes that have a label (i.e. any class that is not the miscellaneous class).

4 Results

All accuracy for each of the trials are shown in Table 1. The first two trials took about 5 minutes to train and the last two trials took 15-20 minutes to train due to data augmentation. It is important to note the change in accuracy when changing the learning rate and adding data augmentation. As important as accuracy is, it's even more important that model runs well on a mobile device. It may be apparent that the first trial isn't the best solution, but when comparing trial three to trial four we will notice that they are almost identical from an accuracy standpoint. For trials three and four I ran the model on the Android Tensorflow example application. I found that the fourth trial's model was slightly better at detecting the right object as opposed to the third, but the increase was not by much. Users can navigate between the map and creating riddles via a navigation drawer (See figure 3). To create a riddle a user would first scan an object and then input some text (the riddle) that is relevant to the scanned object (See figure 4). To solve a riddle, simply select the marker and then select the "Solve" button on the bottom of the map. A camera interface will appear with riddle of the text located at the bottom of the screen. Keep the camera over the object for 3 seconds to solve the riddle (See figure 5).

Table 1: Results

Trial	1	2	3	4
Validation Accuracy	76.8%	80.8%	81.2%	81.1%
Validation Size	5530	5530	5711	6763

5 Conclusion

Overall all this project was a great learning experience. I came with no practical knowledge of Tensorflow and was able to take a pre-trained mobile architecture (MobileNet) and train it to my custom data set. If I had more time I would have liked to fine-tune the model for more accurate results. Due to the inaccuracies with the model, it did hinder the mobile application when it came to detecting objects.

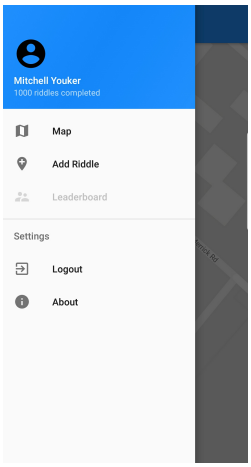
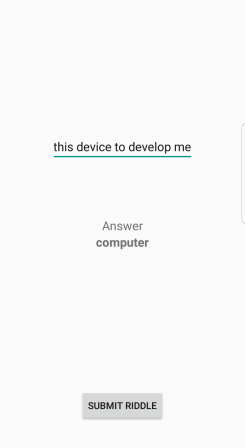


Figure 3: The navigation drawer.

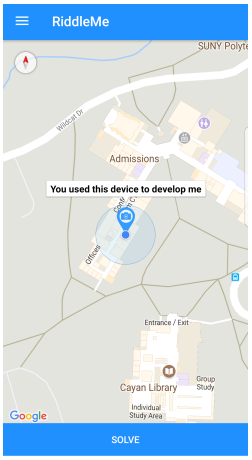
(a) Scan the object.



(b) Inputting the text of the riddle.

Figure 4: Creating a riddle.

I also received firsthand experience on how difficult and fun training models can be when it comes to tinkering with the parameters (learning rate, validation/training set size, data augmentation, etc.) I really did enjoy the Android development aspect to the project. I have had previous knowledge working with iOS applications, but never with Android. I found the ability to drop-in my Tensorflow model into the



(a) Selecting a riddle.

(b) Solving the riddle.

Figure 5: Solving a riddle

example application very helpful for quick testing. I wish that the Android example application was more developed as I have seen they have issues with getting the preview camera to display correctly on certain devices. This project could be expanded in the future, both with more Android features and a better Tensorflow model. Some interesting features that could be implemented are high scores, login, play with friends, etc. As for the Tensorflow model, it would be nice if a larger number of classes could be included.

References

[1] E. Rublee, V. Rabaud, K. Konolige and G. Bradsk *ORB: An efficient alternative to SIFT or SURF*. 2011 International Conference on Computer Vision, Barcelona, 2011

[2] A. Krizhevsky, I. Sutskever and G. Hinton *ImageNet Classification with Deep Convolutional Neural Networks*. Neural Information Processing Systems Conference (NIPS)

[3] TensorFlow Android Camera Demo
`https://github.com/tensorflow/tensorflow/tree/master/te`

[4] A. 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.* CoRR 2017

[5] Tensorflow Retrain
`https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/image_retraining`

[6] Martn Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Rafal Jozefowicz, Yangqing Jia, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Man, Mike Schuster, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Vigas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. *TensorFlow: Large-scale machine learning on heterogeneous systems* 2015. Software available from tensorflow.org.

[7] ImageNet
`http://www.image-net.org`