Efficient Models Selecting

A Major Qualifying Project
Submitted to the Faculty of
Worcester Polytechnic Institute
in partial fulfillment of the requirements for the
Degree in Bachelor of Science
in
Computer Science
By

Ming Shen
Dongsheng Sun
Yang Xu

Date: 3/22/2018
Project Advisor:
Professor Tian Guo, Advisor

This report represents work of WPI undergraduate students submitted to the faculty as evidence of a degree requirement. WPI routinely publishes these reports on its web site without editorial or peer review. For more information about the projects program at WPI, see http://www.wpi.edu/Academics/Projects.
Abstract

The goal of this project is to enhance the efficiency of mobile device machine learning using deep learning pre-trained models. This is achieved by dynamically changing pre-trained model files based on the accuracy and latency of models. Ranking all available models, the application compute the average accuracy and latency after every time usage.
Executive Summary

Nowadays, with the development of machine learning technology, more and more mobile application start to implement machine learning for mobile devices such as Google translation and Apple face recognition. People begin to realize the machine learning could change the way they live. For example, one day, John travelled to Japan but he does not know any Japanese except simple word like hello and sorry. However, he installed Google Translation in his phone which can make people from different country communicate without any trouble, also the application even could be used without any internet connection. There are a lot of examples like this happening currently. Therefore, as the increasing numbers of mobile machine learning application, the efficiency of the application will be crucial.

The goal of this project is to improve the efficiency of mobile machine learning by using different pre-trained models. Efficiency here stands for using the highest accuracy models among all the pre-trained models to do image reference, in other words, we will use the model, which has the highest accuracy of prediction when each image arrives to our android app. Also, apart from the accuracy, the time of each prediction time is also recorded for future use (not in this project). In the process of pre-trained models do inference, data are collected including whether this model does a right prediction or a wrong prediction. When new image comes in, the pre-trained model with the highest accuracy will be chosen to do the inference. If the result is correct, the accuracy of this model would increase, and if it is a wrong prediction, the accuracy will drop. Then next time when a new image comes in, since the accuracy dropped, it may be defeated by another model thus it won’t be used to do prediction again, instead, the model, which defeat this one will do the prediction task. The app goes on in this cycle. From the figure 1 and figure 2, they show the two main activity of the application.
The application contains three part: (1) Frontend structure including android camera activity to invoke system camera and take picture. (2) Database storing test data such as model name, latency and accuracy. (3) Class of Tensorflow API for image recognition. The structure of the application is that once you click on the take picture, it will start the recognition activity that conduct machine learning on the picture that you just took. After you see the result of the recognition and the latency, you can choose if the result is right or wrong, which could insert these information into database for further computation. As the dataset growing, the ranking rate of each model will keep stable. The application will have the most efficient model for future using because this model is going to be the fastest one among all the models bundled with the application without sacrifice accuracy.
The result of this project is that the team has tested about 500 images to see if there is any improvement of inference efficiency. After the test has done, the team plotted a graph to show that showing the latency has decreased with little changing in the accuracy, which indicate that through a great amount of testing, the efficiency has increased.

There are a couple of ways that could optimized the current method. One is that to build a server that store all available open sources pre-trained model and the server can push the latest model to the application. The other is to build a server database to store data from all mobile devices that using this application. Then the server could rank all models in server side and push a couple of good-performance models to client side application.
Table of Contents

Abstract 1

Executive Summary 2

Table of Contents 5

List of Figures 7

List of Tables: 7

1. Introduction 9
   1.1 The Current State of Machine Learning 9
   1.2 The Challenge 9
   1.3 The Solution 9
   1.4 Report organization 11

2. Background 12
   2.1 Deep Learning Pre-trained Models: 12
      2.1.1 MobileNets: 12
      2.1.1 InceptionV3: 13
   2.2 Deep Learning Frameworks: 13
      2.2.1 Caffe: 14
      2.2.2 Tensorflow: 14
   2.3 Mobile Development Backgrounds: 16
      2.3.1 Android SQLite: 16

3. Methodology 18
   3.1 The Algorithm 18
      3.1.1 Input and output 18
      3.1.2 How does it work 18
   3.2 Implementation 19
   3.3 Test Method 21

4. Result 21

5. Related Work 23
   5.1 Deep Spot Cloud: Leveraging Cross-Region GPU Spot Instance for Deep Learning 23
   5.2 TensorFlow 23
   5.3 Google Search 24
   5.4 Google Neural Machine Translation 24

6. Conclusion 25
List of Figures

Figure 1: Demo Camera Activity .................................................................................. 3
Figure 2: Demo Application Result Activity ................................................................. 3
Figure 3: Overall Structure of System .......................................................................... 10
Figure 4: Tensorflow Lite Architecture ........................................................................ 15
Figure 5: SQLite Database Method Structure ............................................................... 17
Figure 6: Android Application Flowchart .................................................................... 20
Figure 7: The results for model # being chosen ......................................................... 22
List of Tables:

Table 1: Table of MobileNet’s Version ........................................................................13
Table 2: Database Schema .......................................................................................19
1. Introduction

1.1 The Current State of Machine Learning

In the recent years, machine learning technologies have become very popular in computer systems. It is widely used for developing software for tasks including computer vision, speech recognition, natural language processing, robot control etc. However, the field of machine learning is still young and is still rapidly expanding, often by inventing new formalizations of machine-learning problems driven by practical applications. Machine-learning systems are increasingly taking the form of complex collections of software that run on large-scale parallel and distributed computing platforms and provide a range of algorithms and services to data analysts [6].

As mobile devices being developed and improved, they have become the platform to execute machine learning tasks like computer vision and voice recognition. The deep learning technologies are frequently implemented in mobile computing systems. A lot of entertaining applications have been developed based on theses technologies.

1.2 The Challenge

Various complicated systems have been developed to solve and computer vision tasks during recent years. The accuracy of the results can be improved by making network deeper. However, the improvement of the accuracy often requires higher computational cost. While in the real world mobile applications that runs on various types of devices with different sets of hardware and software condition, the trade off between accuracy and efficiency needs to be carefully considered.

1.3 The Solution

The overall goal of this project is to come up with an effective solution that can help reduce the time and space cost for accomplishing machine learning tasks on mobile devices without losing accuracy. Thus we developed an algorithm that allow mobile application dynamically
choose the most suitable machine learning model for object detection task. An overall structure of our solution is demonstrated in figure 1.

Figure 3: Overall structure of the system

In order to test the algorithm, we developed a prototype android application that can identify the objects on the photos that users took. The application is designed based on mobile application demo provided in Tensorflow codebase. The application uses Tensorflow Object Detection API to inference models on mobile devices. As the trade-off between time cost and accuracy is being concerned, the algorithm dynamically selects the most suitable machine learning model from all available models. The application will evaluate the effectiveness of different models based on the evaluation of accuracy and execution latency.

In order to test the effectiveness of our solution, we simulated the inferencing process for the application on a Google Nexus 5 smartphone. The testing result demonstrates satisfied accuracy of and efficiency of the image detecting tasks, and the expected ranking of effectiveness for different models is being correctly prioritized. More details about the system design, the implementation, and test methodology will be illustrate in the next sections.
1.4 Report organization

This report is broken into six chapters. The first chapter will firstly introduce the motivation of the project and then provide an overview of the entire project; The second chapter will cover detailed background information for the topics involved during the development in this project. The third chapter will discuss the detail of developing the project and the method we used to evaluate the result; Then the fourth chapter will describe the result which shows the performance and its improvement compared to the existing solutions. More discussion on the results and the insights derived from this project will be stated in the fifth chapter. And the last chapter will discuss the future work we hope to accomplish and the conclusion for the entire project. We appreciate the previous researches that researchers had been working hardly on for years which gave us a significant guidance during the development of the project. And the citations will be listed at the end of the report.
2. Background

2.1 Deep Learning Pre-trained Models:

Deep learning models are loosely related to information processing and communication patterns in a biological nervous system, such as neural coding that attempts to define a relationship between various stimuli and associated neuronal responses in the brain. Some of pre-trained models are for mobile, because most pre-trained models are trained and designed for server side usage which has faster computing speed and more advanced GPU to conduct inference on the target by using large model file size. Comparing to the server side computing, mobile devices do not have powerful CPU and GPU to conduct such work on the large model file. Mobile devices will take much longer time than server station that has state-of-the-art GPU. Another reason is that when the application is installing, it is bundled with all model files, which is going to take up a large storage space of the device. Currently, there are a lot of different kinds of pre-trained models that designed for mobile device. For example: MobileNets models, InceptionV3 and etc.

2.1.1 MobileNets:

MobileNets is designed to effectively maximize accuracy while being mindful of the restricted resources for an on-device or embedded application[7]. The MobileNets Models are small, low-latency, low-power, which are parameterized to meet the resource constraints of a variety of use cases. They can be built upon for classification, detection, embeddings and segmentation similar to how other popular large scale models, such as Inception, are used. MobileNets can be run efficiently on mobile devices with TensorFlow Mobile. MobileNets trade of between latency, size and accuracy while comparing favorably with popular models from the literature. According to the figure below, it indicates that along with the version advancing, more MACs and Parameters are added to MobileNet, which increases the general accuracy.
### Table 1: Table of MobileNet’s Version[7]

<table>
<thead>
<tr>
<th>Model Checkpoint</th>
<th>Million MACs</th>
<th>Million Parameters</th>
<th>Top-1 Accuracy</th>
<th>Top-5 Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>MobileNet_v1_1.0_224</td>
<td>569</td>
<td>4.24</td>
<td>70.7</td>
<td>89.5</td>
</tr>
<tr>
<td>MobileNet_v1_1.0_192</td>
<td>418</td>
<td>4.24</td>
<td>69.3</td>
<td>88.9</td>
</tr>
<tr>
<td>MobileNet_v1_1.0_160</td>
<td>291</td>
<td>4.24</td>
<td>67.2</td>
<td>87.5</td>
</tr>
<tr>
<td>MobileNet_v1_1.0_128</td>
<td>186</td>
<td>4.24</td>
<td>64.1</td>
<td>85.3</td>
</tr>
<tr>
<td>MobileNet_v1_0.75_224</td>
<td>317</td>
<td>2.59</td>
<td>68.4</td>
<td>88.2</td>
</tr>
<tr>
<td>MobileNet_v1_0.75_192</td>
<td>233</td>
<td>2.59</td>
<td>67.4</td>
<td>87.3</td>
</tr>
<tr>
<td>MobileNet_v1_0.75_160</td>
<td>162</td>
<td>2.59</td>
<td>65.2</td>
<td>86.1</td>
</tr>
<tr>
<td>MobileNet_v1_0.75_128</td>
<td>104</td>
<td>2.59</td>
<td>61.8</td>
<td>83.6</td>
</tr>
<tr>
<td>MobileNet_v1_0.50_224</td>
<td>150</td>
<td>1.34</td>
<td>64.0</td>
<td>85.4</td>
</tr>
<tr>
<td>MobileNet_v1_0.50_192</td>
<td>110</td>
<td>1.34</td>
<td>62.1</td>
<td>84.0</td>
</tr>
<tr>
<td>MobileNet_v1_0.50_160</td>
<td>77</td>
<td>1.34</td>
<td>59.9</td>
<td>82.5</td>
</tr>
<tr>
<td>MobileNet_v1_0.50_128</td>
<td>49</td>
<td>1.34</td>
<td>56.2</td>
<td>79.6</td>
</tr>
<tr>
<td>MobileNet_v1_0.25_224</td>
<td>41</td>
<td>0.47</td>
<td>50.6</td>
<td>75.0</td>
</tr>
<tr>
<td>MobileNet_v1_0.25_192</td>
<td>34</td>
<td>0.47</td>
<td>49.0</td>
<td>73.6</td>
</tr>
<tr>
<td>MobileNet_v1_0.25_160</td>
<td>21</td>
<td>0.47</td>
<td>46.0</td>
<td>70.7</td>
</tr>
<tr>
<td>MobileNet_v1_0.25_128</td>
<td>14</td>
<td>0.47</td>
<td>41.3</td>
<td>66.2</td>
</tr>
</tbody>
</table>

#### 2.1.1 InceptionV3:

Inception is derived from ImageNets but with less computational cost and small model size[8]. However, the accuracy of InceptionV3 does not decrease with the low computational cost. Inception-v3 is trained for the ImageNet Large Visual Recognition Challenge using the data from 2012. The models is designed to classify entire images into 1000 classes in this challenge. To compare models, the TensorFlow team examine how often the model fails to predict the correct answer as one of their top 5 guesses. Inception-v3 reaches 3.46%, while AlexNet achieved 15.3% and Inception(GoogLeNet) achieved 6.67%[9].

#### 2.2 Deep Learning Frameworks:

Frameworks are the tools that help artificial intelligence and machine learning grow up in fast speed. They can build the deep learning solution easily because of the higher level of abstraction and simplify potentially difficult programming task[19]. There are tens of open sources deep
learning frameworks offering different purposes for different users. Some are designed for researches purposes and others are built for application purposes.

2.2.1 Caffe:

Caffe is a deep learning framework made with expression, speed, and modularity in mind. It is developed by Berkeley AI Research (BAIR). Caffe is a open source deep learning framework. It has four features that make Caffe useful in deep learning area[10]. Caffe provide Caffe supports many different types of deep learning architectures geared towards image classification and image segmentation. It supports CNN, RCNN, LSTM and fully connected neural network designs. Caffe supports GPU-based acceleration using CuDNN of Nvidia.

2.2.2 Tensorflow:

TensorFlow is an open source software library for numerical computation using data flow graphs. Nodes in the graph represent mathematical operations, while the graph edges represent the multidimensional data arrays (tensors) communicated between them. The flexible architecture allows you to deploy computation to one or more CPUs or GPUs in a desktop, server, or mobile device with a single API. TensorFlow was originally developed by researchers and engineers working on the Google Brain Team within Google’s Machine Intelligence research organization for the purposes of conducting machine learning and deep neural networks research, but the system is general enough to be applicable in a wide variety of other domains as well[11].

TensorFlow Lite is the team that used to program the Android image recognition application. It is TensorFlow’s lightweight solution for mobile and embedded devices. It enables on-device machine learning inference with low latency and a small binary size. To achieve low latency on image recognition, TensorFlow Lite provided three main solutions: optimizing the kernels for mobile apps; pre-fused activations; and quantized kernels that allow smaller and faster(fixed-point math) models[12]. From the figure below, the architecture shows that in the tensorflow project, there is a tensorflow lite converter to convert the regular trained tensorflow models to light models which could fit properly on the mobile devices.
TensorFlow Lite is even more powerful on Android devices. It provides Android Neural Networks API library for better interface between device to leverage hardware acceleration[16].

The advantage of TensorFlow Lite:

- Android Neural Networks API and C++ API
- Flexible Model File ended with .tflite
- Speed: New Neural Network API makes computation much faster.
- Privacy: The data does not leave the device.
- Availability: it could be used without any internet connection.
- No Computation Cost: All the computation is performed on your device. So there is no cost for the computation on server.
- Pre-tested models: All the models work out of the box. For example, Inception V3, MobileNets
- Cross Platform: A runtime designed to run on many different platforms, starting with Android and iOS[14].

The trade-offs:

- System-utilization: using neural networks involve a lot of computation, which will use more battery power than normal.
- **Application size:** The model used in image recognition could take up multiple megabytes of space. If bundling those large models with the APK would cause bad user experience, like cost a lot of cellular data when user use the application[15].

2.3 Mobile Development Backgrounds:

As the growth of machine learning industry, people can access this technology in different aspects, especially in their daily life with their smartphones. For example, people can use their phones to recognize objects or use translation application to communicate with other people that speaking total different languages. Designing and developing mobile applications with machine learning features is going to be a trend.

2.3.1 Android SQLite:

The team choose to use the built-in SQLite as the database to store image information and recognition results. The SQLite is a open-source SQL database that stores data to a text file on a device. SQLite supports all the relational database features. To access SQLite in Android, Android SQLite provide same syntax as regular SQLite like Insert, Update, Query and Delete by writing query sentence in Java. According to the structure graph below, it shows that the SQLite
Database Object provide these abstraction method that can modify the database.

Figure 5: SQLite Database Method Structure[17]
3. Methodology

3.1 The Algorithm

The main goal of the algorithm is to find the most suitable machine learning model based on the performance of the models when they were used to inference user input on a certain device. The performance of object detection models varies according to the types of objects being detected, the quality of input graphics and many other factors. In this project, we developed an algorithm that can dynamically select models from the model base.

3.1.1 Input and output

As previously stated, we rank the effectiveness of different algorithms by their accuracy. There are various ways to evaluate the precision of the models depends on the type of the machine learning task. For object detection, an object detection precision matrix is always used during the training phase. However, the real world object detection tasks are always different from the training dataset and the image being inferred are always unlabeled. In such case, the precision matrix can be simplified to be the accuracy of each detected objects. The overall accuracy of each model can be calculated by averaging historical results. Thus, the algorithm consumes precision value for each inference. And the model with the highest precision will always be produced.

3.1.2 How does it work

Let’s say we have $n$ available models, being marked from $M_1$ to $M_n$, with current average precision $P_1$ to $P_n$. Assume $M_1$ currently has highest precision, we have $P_1 = \text{MAX}(P_1...P_n)$. Noticing that $P_1$ is calculated by averaging precision of inferences when $M_1$ was used, which can be annotated as $\sum_{i=0}^{X} p_i$ where $i$ represents the time that the model is being used. When $M_1$ is being repeatedly used in a certain scenario, its average precision will get updated. Once $M_1$ no longer has the highest accuracy at some point, the algorithm will select a new model to execute the upcoming task.
3.2 Implementation

The ideal situation for a mobile application with machine learning should be low latency and high accuracy, while also maintaining good battery performance. However, the high accuracy is often the trade-off for low latency and battery consumption. Many applications have high latency in order to get better accuracy on the result. For instance, an embedded speech recognition system that runs locally on a mobile device is more reliable and can have lower latency; however, it must be accurate and must not consume significant memory or computational resources[18].

This project is designed to enhance the time of deep learning by implementing optimization algorithms without sacrificing the result accuracy. Since the different models could cause different performance on the same target, the team decided to improve the performance by dynamically choosing the model for the target based on the result collected on every use, which means the rating for models are changing as the application being used every time. To achieve this goal, the team modified the example Android TensorFlow Lite project.

The database was designed to record the prediction of each image. Every time an image is predicted, a row of data is inserted into the database. Here is the schema of the database:

<table>
<thead>
<tr>
<th>Name</th>
<th>Accuracy</th>
<th>Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>First data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Second data</td>
<td></td>
<td></td>
</tr>
<tr>
<td>...</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table 2: Database Schema

Name stands for the name of the model and in code it is a string type. In our project, we have three models and for simplicity, we just call them “model1”, “model2”, and “model 3”. Accuracy is either 1 or 0 where 1 stands for the correct prediction in this row of record and 0 stands for incorrect prediction in this row. Time stands for the time period the prediction takes.
First of all, when the database does not contain enough information to let the application choose proper model, the application will randomly choose model for the image. For example, assuming there are three models, the team set rate of using to 0 for each of them. So when the next object coming in, the application will randomly choose a model for it. For the further manipulation, the application will store average process time and average accuracy to the database after calculation. The database will contain the file locations of all available models, the average latency of them and the accuracy of them. Therefore, when the next object is coming in, the application will rank all available models for the next object based on the combination of new average time and new average accuracy.

At last, after thousands of and millions of time usages, the application can choose proper model based on the data has been collected for most of objects.
3.3 Test Method

In order to test the effectiveness of our algorithm, we developed a benchmark android application to see if the algorithm can dynamically choose model under different scenario. We chose three pre-trained models from Tensorflow object detection model zoo, the initial models are all trained use coco dataset [20].

The benchmark app will be able to record and store the inference precision and response time into a SQLite database. For the first 100 inferences, we assign random model to finish the tasks. This will generate initial values for average precision, disregarding weather the best model was chosen. Since the goal of the algorithm is to dynamically choose the current best model for certain series of tasks, we simulate the next 400 inferences to see if the most precise model will be bring to the top priority, which help us prove the correctness of the algorithm.

4. Result

As described in section 3.3, we conducted tests on a Google Nexus 5 mobile phone, with Android 7.0 as its operating system. We simulated 500 inferences, and the model chosen by the algorithm is demonstrated on Figure 6. Notice that for the first 100 execution, the model is randomly chosen without regards to the precision. In this experiment, we used ssd_mobilenet_v1 as model #1, faster_rcnn_inception_v2 as model #2 and faster_rcnn_resnet50 as model#3. The models are all checked out from Tensorflow detection model zoo.
The result showed that when the average precision become normalized, model #1 outcompetes the other two models and become the recommended model by the algorithm. Notice that between trial 100 and 120, the preferred model was once switched between model #1 and #2, which was the time when precision of model #1 and model #2 were very close. However, the data from the database shows that the average precision of model #2 had dropped a bit and become steady a few percent under #1.
5. Related Work

5.1 Deep Spot Cloud: Leveraging Cross-Region GPU Spot Instance for Deep Learning

Cloud computing resources that are equipped with GPU devices are widely used for applications that require extensive parallelism, such as deep learning. When the demand of cloud computing instance is low, the surplus of resources is provided at a lower price in the form of spot instance by AWS EC2. This paper proposes Deep Spot Cloud that utilizes GPU equipped spot instances to run deep learning tasks in a cost efficient and fault-tolerant way.

The main difference between this related research and ours is that this related research focus on utilizing instances in different regions across continents as a single resource pool to deal with the price dynamicity of the GPU spot instance, and it also proposed proposes a task migration heuristic by utilizing a checkpointing mechanism of existing deep learning analysis platform to conduct fast task migration when a running spot instance is interrupted. However, we mainly focus on pursuing better performance efficiency and inferencing accuracy on android device.

5.2 TensorFlow

TensorFlow was designed to be a good deep learning solution for mobile platforms including Android and iOS. Currently we have two solutions for deploying machine learning applications on mobile and embedded devices: TensorFlow for Mobile and TensorFlow Lite. TensorFlow was designed from the ground up to be a good deep learning solution for mobile platforms like Android and iOS, and TensorFlow Lite is TensorFlow lightweight solution for mobile and embedded devices, which enables on-device machine learning inference with low latency and a small binary size. Our research is based on the TensorFlow for Android because we are using TensorFlow model to do inference on photos taken by our apps.
5.3 Google Search

Google search is a web search engine developed by Google and it is the most-used search engine on the world wide web. The order of search results returned by Google is based, in part, on a priority rank system called “PageRank” and the main purpose of Google Search is to hunt for text in publicly accessible documents offered by web servers, as opposed to other data, such as images or data contained in database. Machine learning is highly connected to Google Search since the main algorithm of searching is based on machine learning. In general, a few things should know about Google Search: pattern detection, identifying new signals, custom signals based on specific query, image search to understand photos, identifying similarities between words in a search query, improving Ad quality and Targeting for users, synonyms identification, and query clarification.
Relation between Google Search and our project is that we both use machine learning, but we are trying to leveraging on deciding the best model to use when doing image recognition, however, Google Search focuses on collecting the data over the world wide servers.

5.4 Google Neural Machine Translation

Google Neural Machine Translation (GNMT) is a neural machine translation (NMT) system developed by Google that uses an artificial neural network to increase both fluency and accuracy. The way GNMT improves on the quality of translation is by “learns from millions of examples” where example based machine translation method (EBMT) is used. More importantly, the GNMT network can undertake interlingual machine translation by encoding the semantics of the sentence, rather than by memorizing phrase-to-phrase translation.
This technique maybe is not directly related to our project, however, both of our projects are trying to do prediction on some data using machine learning.
6. Conclusion

As demonstrated in section 4, the test result has shown the algorithm could effectively finds the most suitable machine learning model when an object detection task is being requested. However, when the sizes of the machine learning models and the latency of loading the model before inferencing are being concerned, there are still many aspects we can think about to optimize and improve the effectiveness of the application. This project gave us insights about different ways to improve the performance of machine learning when it is brought to small scale devices like smartphones. We are looking forward to discover more about improving the effectiveness of machine learning and we hope to find more solutions when our focuses move from end-devices only to a more sophisticated computing system.
7. Reference


