Static Energy Consumption Analysis for Android Applications

Zhimin Dai¹, Wei Wang*¹ and Yi Wu¹

¹Department of Software, Yunnan University, Kunming, Yunnan, 650500, China.
*Corresponding author’s e-mail: wangwei@ynu.edu.cn

Abstract. Energy consumption of mobile application software is an important factor influencing the battery life of smart phone terminals. Currently, for the tens of thousands of Android software, we propose a static visual detection and classification method based on application energy consumption. Compared with the high-precision and complex application component energy consumption model, this model analyzes the bytecode images generated by the dex and xml files in the Android apk and trains them with deep learning methods. The energy consumption level of two kinds of different mobile terminal software can be obtained, which can quickly estimate the energy consumption of mobile terminal when the application is running. Experimental results show that the model's classification accuracy of energy consumption estimation is 50.49%, which can help mobile end users to easily predict the battery power consumed by applications.

1. Introduction

Nowadays, the analysis of application energy consumption of smart phones is almost accompanied by the whole development process of smart phones while people are enjoying the audio-visual benefits of smartphones, mobile apps are also consuming a lot of power. With the cooperation of hardware and software, researchers pay more and more attention to the application level energy consumption analysis of mobile phones. The current Google play App market contains tens of millions of android apps, and App Annie, a Mobile data and analytics provider, said in its latest "The State of Mobile 2019" report that App downloads worldwide exceeded 193 billion in 2018, up 35% from 2016. In such a large number of downloads, many apps do not adopt effective software energy saving methods, so it is particularly important to provide users with detecting apps and estimating app energy consumption grade classification methods.

In previous energy consumption analysis of mobile applications, it can be seen that literature calculates the CPU energy consumption of Android applications by repeating the execution of 20 million times for each CPU instruction and measuring its energy consumption with an external instrument, thus obtaining the CPU energy consumption of one execution of the instruction. The method of building the power consumption model of each hardware of mobile phone off-line is introduced in the literature. The research work of literature et al. used "BMU" to read the discharge voltage and current of mobile phone batteries. The structural accuracy of the energy consumption model of Android system affects the accuracy of these Android energy consumption measurement techniques. This paper presents a new static detection method. Namely, the visualization method of static energy consumption level of android APP and the classification of unknown installation packages. The Dalvik bytecode in apk was transformed into a grayscale image, and the features were classified by combining with the deep learning method.
2. Data Preprocessing

2.1 Classification of Energy Consumption Grade
This paper adopts dichotomy to classify the energy consumption of android apps into high and low energy consumption. PowerTutor method is used to obtain the android applications for energy consumption, crawl Google Play app store application software installation package, to conduct stress tests in the android virtual machine, Google automation test framework MonkeyEnergyTest men in the study, the framework can be implemented for app install package bulk installation and simulate artificial mobile phones operation such as: installation, sliding, click, unloading operation. In addition to the energy consumption data including power data and wakelock data, the secondary PowerTutor open source tool also collects the time of a single application during the stress test, as well as the power data per unit of time when running in the background[4]. The application software for energy assessment is set as S, and the application software set is \( S = \{ S_1, S_2 \} \). Among them, \( S_1 = \{ a_1, a_2, ..., a_n \} \), \( S_2 = \{ b_1, b_2, ..., b_n \} \). During the test, after each app is tested, the corresponding electric quantity data E is output, and the highest energy consumption value is \( E_{\text{max}} \), take the lowest energy consumption \( E_{\text{min}} \). So the middle number is \( E_{\text{mid}} = \frac{E_{\text{max}} + E_{\text{min}}}{2} \). The threshold value is \( E_{\text{mid}} \). Is greater than \( E_{\text{mid}} \) energy consumption is \( S_1 \), the high energy consumption, lower than \( E_{\text{mid}} \) energy consumption is \( S_2 \), that is, low energy consumption.

2.2 Visual processing of APK
The APK file of Android application is compiled by the Android SDK, and all the data and resources are packaged into a file with the suffix APK (Android package). The number of image visualization researchers on the Android code is less than that on the PC platform. Because the current Android applications mostly use tools for code before packaging APK confusion and polymorphic deformation, greatly enhances the Android application art to the engineering ability, the malicious code for the Android platform static detection has brought great challenges, but through this chapter puts forward the method, only need to unzip the APK package without decompiling, can solve this problem. Extract the code file after unzipping APK, and store each 8-bit binary number in the array into a value for the previously unzipping file classes.dex in the range of 0-255, corresponding to the 256-order grayscale[5]. The resulting grayscale image file resolution is related to the size of APK, so the grayscale image generated uniformly is a square and scaled to a uniform size.

![Figure 1. A grayscale image composed of class.dex and androidmanifest.xml.](image)

The classes.dex file structure obtained by unpacking the APK package is composed of several structures, including dex_header, field_ids, string_ids, type_ids, class_def, proto_ids, method_ids, data,
The classes.dex file header records some properties of the dex file and the rest of the data structure and relative offset positions. For the extracted file androidmanifest.xml, the image is generated in the same way as classes.dex. However, since the androidmanifest.xml file is generally small, the image size is 512*10, and the grayscale image of 512*512 is formed under the image generated by classes.dex[9], which is convenient for the input processing of a sample. After the input into the network, a convolutional neural network is constructed for the two parts, and the penultimate full connection layer of the network is taken as the output feature. After splicing, the model is fused[10].

![Gray scale image samples of two apks with different energy consumption.](image)

Figure 2. Gray scale image samples of two apks with different energy consumption.

Figure 2 shows the randomly selected grayscale image samples of APK with two different energy consumption, high energy consumption $S_1$ and low energy consumption $S_2$. For the four apk-generated grayscale images, the grayscale images with different energy consumption have some differences in texture, so the human eye recognition ability is not strong. However, as a strong and effective image classifier, CNN can identify these gaps, learn different features and classify them.

3. Construction of convolution network

3.1 Construction of energy consumption detection based on ResNet network

As a powerful residual network under CNN, ResNet network refers to the VGG19 network and is modified on its basis. In the depth structure of CNN, gradient dissipation and gradient explosion will occur as the network layer goes deeper, resulting in a decline in the accuracy of the training set[7]. This problem can be solved by residual network, so that the performance of the network can be improved at the same time.

![ResNet network](image)

ResNet using a new connection method called Shortcut connection, which adds a new Identity mapping to the original network structure, allows you to learn the function you originally needed.
$F(x)$ convert $F(x) + x$. That is, the sum of the learning function and input data[8]. This simple addition will not add extra parameters and computation to the network, but can improve the training speed and effect of the model. When the number of layers of the model is deepened, this structure can well solve the degradation problem. This study is based on the ResNet structure in the network's input is $512 \times 512$ gray image, using 16 convolution layer, according to ResNet model structure, 16 convolution layer is divided into five groups, the size of the convolution kernels respectively $1 \times 1$, $3 \times 3$, $1 \times 1$, used in the characteristics of the input image are extracted, convolution layer activation function is nonlinear Relu activation function, the biggest pooling layer using a $3 \times 3$ (Max Pool) and $7 \times 7$ Average pooling layer (business Pool), step length set to 2, so the aspect to $1/2$, Finally, two full connection layers and Softmax activation functions are used to classify the grayscale images composed of APK class.dex and androidmanifest.xml.

Figure 4. network structure diagram.

In figure 4, there are two kinds of lines between convolution layers: dotted lines and solid lines. The connection part of the solid line performs the convolution of $3 \times 3 \times 64$, and the number of channels is the same, so the output result $y$ According to formula (1), the connected part of the dotted line is the convolution operation of $3 \times 3 \times 64$ and $3 \times 3 \times 128$, respectively. The number of channels is different, namely 64 and 128, and the output result is obtained. The calculation method of equation (2) is adopted. Among them, $y$ is the output result of the model, $F(x)$ is the function to be learned, $x$ is the input value of the model, $W$ is the convolution, operation to adjust the $x$ channel dimension.

\begin{align}
    y &= F(x) + x \quad \text{(1)} \\
    y &= F(x) + W \ast x \quad \text{(2)}
\end{align}

4. Experiment and analysis

4.1 Experimental environment and evaluation parameters

The machine with NVIDIA GeForce GTX 1080Ti graphics card and Ubuntu 16.04 operating system were selected. The Keras deep learning framework was selected for model training and testing. The
classification model based on ResNet deep network was tested on the APK grayscale image data set after splicing processing. The data set 6000 different APK grayscale images as the training set and 600 images as the test data. During the model training, the learning rate was set to 0.001, and the training rounds were 50 times. The APK gray image energy consumption classification based on the ResNet deep network was trained, and the model parameters obtained from the training were saved for subsequent testing and prediction. Then the trained model is tested on the data to obtain the accuracy and loss of the model. After the training of deep learning model is completed, its accuracy and loss value need to be evaluated. The loss function in this system is calculated using the logarithmic loss function (logistic regression), and the calculation process is as follows [11]:

\[
J(\theta) = -\frac{1}{m} \sum_{i=1}^{m} [y^{(i)} \log h(\theta(x^{(i)})) + (1 - y^{(i)}) \log (1 - h(\theta(x^{(i)})))]
\]  

(3)

The formula for calculating the accuracy is:

\[
Accuracy = \frac{TP + TN}{TP + TN + FP + FN}
\]  

(4)

In formula (4), TP represents the number of positive classes judged as positive classes, FP represents the number of negative classes judged as positive classes, FN represents the number of positive classes judged as negative classes, and TN represents the number of negative classes judged as negative classes.

4.2 Softmax classifier

The convolutional neural network adopts softmax classifier, and the softmax layer is connected to the full connection layer. The output of softmax layer is a probability distribution, and it can be obtained that the input belongs to the probability of each category, and the sum of the output probabilities is 1. The output probability is calculated as formula [12]:

\[
f_\theta(x_i) = \begin{bmatrix}
P(y_i = 1|x_i; \theta) \\
P(y_i = 2|x_i; \theta) \\
\vdots \\
P(y_i = k|x_i; \theta)
\end{bmatrix} = \frac{1}{\sum_{j=1}^{k} e^{f_j x_i}} e_{y_i}^{T} x_i
\]  

(5)

In the equation, the value of k is 2 (this study is a dichotomy problem, so k=2).

4.3 Evaluation of different network models

Since different network models and complexity would affect the training results, three different network models were selected for research and compared under the same number of training rounds. The comparison results of training are shown in table 1.

| Trainmodel | Traindata | Testdata |
|------------|-----------|----------|
| CNN        | 0.3853    | 0.3291   |
| VGG-16     | 0.4533    | 0.4122   |
| RestNet-16 | 0.5049    | 0.4206   |

Table 1 shows the three different models to the final accuracy after 50 iterations and loss value, and by this article can be seen in table 1 ResNet convolution adopted by the neural network model in the 50 rounds a smaller number of iterations to "learn" the APK energy consumption characteristic of the gray image, in the training set and validation set accuracy reached 50.49% and 42.06% respectively, and the loss of value of 6.0440 and 7.2340, this is a good shows ResNet convolution neural network model can classify the static gray image generated by the APK, Whether the APK is high energy consuming software.
5. Conclusion
In this paper, a static APK grayscale image classification method based on ResNet deep network was implemented, and two different APK grayscale images with different energy consumption were classified. The experimental results show that although the accuracy is slightly lower, in order to improve this situation, it is necessary to increase the data amount of high-energy and low-energy application software, and to extract more abundant characteristic values for the special images such as bytecode images, and the influence of changes in the training model and the improvement of training methods on the training results. It is hoped that the future work can improve the experimental results, further optimize the energy consumption level of Android application software, improve system performance and improve user experience. Secondly, the principle of bytecode image technology is to convert the binary file into an array of pixels corresponding to the grayscale image, and obtain the texture features of the software according to the generated grayscale image. This technique can not only obtain the characteristics of DEX files, but also be applied to binary files such as EXE, cclass.dex and XML, providing a way to obtain the characteristics of closed-source software. Therefore, the method can be applied to software areas other than Android applications. Although this study could not achieve a high accuracy rate in the final results, it provided a new static method to detect APK energy consumption, which is believed to further improve the experimental results in the following studies.

References
[1] Zhang L, Tiwana B, Dick R P, et al. Accurate online power estimation and automatic battery behavior based power model generation for smartphones[C]. International conference on hardware/software codesign and system synthesis, 2010:105-114.
[2] Patil P S, Doshi J, Ambawade D. Reducing power consumption of smart device by proper management of WakeLocks[C]. Advance Computing Conference, 2015: 883-887.
[3] Tiwari V, Malik S, Wolfe A. Power analysis of embedded software: a first step towards software power minimization [J]. IEEE Trans. very Large Scale Integr. syst, 1994, 2(4): 437-445.
[4] Zhang J, Musa A; Le W. A comparison of energy bugs for smartphone platforms[C]. Engineering of Mobile-Enabled Systems, 2013: 25-30.
[5] LeconY, Bottou I, Bengio Y, et al. Gradient-based learning applied to document recognition[J]. Proceedings of the IEEE, 1998, 86(11): 2278-2324.
[6] Srivastava N, Hinton G, Krizhevsky A, et al. Dropout: a simple way to prevent neural networks from overfitting [J]. Journal of Machine Learning Research, 2014, 15(1): 1929-1958.
[7] Simonyan K, Zisserman A. Very Deep Convolutional Networks for Large-Scale Image Recognition[J]. Computer Science, 2014.
[8] Zhang L, Gordon M S, Dick R P, et al. ADEL: an automatic detector of energy leaks for smartphone applications[C]. Eighthleee/acm/ifiplnternational Conference on Hardware/software Codesign and System Synthesis, 2012: 363-372.
[9] Vekris P, Jhala R, Lerner S, et al. Towards verifying android apps for the absence of no-sleep energy bugs[C]. Usenbx Conference on Power-Aware Computing and Systems, 2013:3-3.
[10] Mirzaei N, Malek S, Esfahani N, et al. Testing android apps through symbolic execution[J]. Acm Sigsoft Software Engineering Notes, 2012, 37(6):1-5.
[11] Krizhevsky A7 Sutskever I, Hinton G E. ImageNet classification with deep convolutional neural networks[C]. International Conference on Neural Information Processing Systems, 2012:1097-1105.
[12] Lecun Y, Bengio Y, Hinton G. Deep Learning[J]. Nature, 2015, 521(7553):436.