Title Text

Subtitle Text, if any

Name1
Affiliation1
Email1

Name2 Name3
Affiliation2/3
Email2/3

Abstract

Categories and Subject Descriptors CR-number [*subcategory*]: third-level

General Terms term1, term2

Keywords keyword1, keyword2

1. Introduction

In the recent decade, deep neural networks have a prominent performance in many high-impact machine learning applications. These include speech recognition (Hinton et al. 2012),object classification (Krizhevsky et al. 2012; Sermanet et al. 2013),image caption generation (Vinyals et al. 2015; Karpathy and Fei-Fei 2015) and semantic segmentation (Long et al. 2015; Dai et al. 2015; Hariharan et al. 2015). As the size of data sets is increasingly large, so do the number of parameters in deep neural networks for the sake of comprehending the enormous amount of useful information contained in data sets, such as some modern networks like AlexNet (Krizhevsky et al. 2012)(60M parameters), and VGG-16 (Simonyan and Zisserman 2014)(130M parameters).

Large DNN models are very powerful but energy-consumed and time-consuming, especially during training (Krizhevsky et al. 2012; Simonyan and Zisserman 2014). Thus training these network on wearable, smartphones and Internet-of-Things devices are impractical. As a result, prominent examples of deep learning applied on smart-phones(e.g.,speech recognition) are primarily cloud-supported. Hence increasingly, these networks are trained on industrial-sized clusters (Le 2013) or high-performance graphics processing units(GPUs) (Catanzaro 2013; Hauswald et al. 2015). What's more, in the actual industry, there is a significant class of emerging internet services known as intelligent personal assistants(IPAs), like Apple Siri, Google Now, Microsoft Cortana and Amazon Echo. However, the cloud-assisted pattern introduces important negative side-effects: 1) it raises an issue with regard to user privacy (Harris 2015) as some sensitive data (e.g.,audio or video) is processed in the cloud by a third party; and 2)it only works when sufficient bandwidth is feasible and suffers artificial delays through network traffic (Kosner)(i.e., latency,throughput).

To solve the above problems, several kinds of emerging solutions have been proposed. A kind of solutions is to directly train small models for the on-device classification; however, these tend to observably impact accuracy (Chun and Maniatis 2009), leading to customer discontent. Another kind of solutions is to compress pre-trained deep networks, which have been trained on industrial-sized clusters(server-side). More and more researchers are studying in this direction. Recent work by Denil et.al. (Denil et al. 2013) proves that there exists a amazingly large amount of redundancy among the weights of deep neural networks. The authors point out that a slight amounts of the weights are adequate to reconstruct the entire network. Accordingly, through pruning the redundant, non-informative weights, the size of large trained neural networks can be reduced up to 50X (Han et al. 2015b,a) without any loss of accuracy. More recently, other various methods based on low-rank decomposition (Denton et al. 2014), vector quantization (Gong et al. 2014), hashing techniques (Chen et al. 2015), circulant projection (Cheng et al. 2015), tensor train decomposition (Novikov et al. 2015), and network binarization (Courbariaux et al. 2014, 2015) were proposed without significant drop in the prediction accuracy.

In a word, Compression is an effective way to transfer clumsy neural networks from server-side into mobile-side. A obvious advantage here is that these compression schemes may actualize local processing and storage of neural networks, and to some extent, protect user privacy as well as avoid the response delays and energy consumption from bandwidth restrictions. However, in these pattern, the neural networks deployed on mobile devices is fixed and can not relearn against reality.

As increasing works have successfully achieve the application of deep neural networks without compression on mobile devices (Latifi Oskouei et al. 2016; Oskouei et al. 2015; Brändle et al. 2015; Yanai et al. 2016; Mittal et al. 2016; Zhang). It is imperative to further enhance the interaction with the mobile end user. For example,

In this paper, we propose a concept of application frameworks

Overall, this paper makes the following contributions.

As the most representative networks like AlexNet(240MB), and VGG-16(520MB), are too large to deploy on mobile devices and embedded systems.

Deploying convolutional neural networks(CNNs) for various intelligent tasks on embedded and mobile devices(e.g., smartphones) is obtaining more and more attention.

2. Background And Motivation

2.1 Background

2.2 Personalized Features

In this paper, we consider five kinds of contextual information as follows:

- L**:**
- D, H**:**

Table 1: Notation used in this paper.

Symbol	Description
L	the location
D	day of week
H	hour of day
Wf	the rate level of wifi
B	the battery level
C	a flag whether the battery is being charged
LP	the latest used App
A	the App
CPD	the conditional probability distribution

- Wf:
- B, C:
- LP:

Table 1 summarizes the features considered. What's more, the table also give the description of acronyms to be used in the following passage.

2.3 Traditional Bayesian Algorithm

The traditional Bayesian algorithm is shown in Equation 1, we calculate each App's prediction score given the context by multiplying the CPDs(the conditional probability distribution) of the App and each context. Then the top scored Apps are the final predictions. The CPD $P(A|C)$ denote that the probablity of launching an App under the condition of the last used App. The meaning of the other three CPDs is similar.

$$Socre(A) = P(A|C) \times P(A|D,H,L) \times P(A|W) \times P(A|U) \quad (1)$$

It is intuitive that we should consider as many records (i.e., training data) as possible in order to achieve high prediction accuracy. Because the penalty for using small training sets is over fitting: while in-sample performance may be excellent, The previous work on App prediction has tended to make use of a lot of training samples. However, we need to take the specific area of application into account. Using machine learning algorithm to predict next App that smartphone user will be used, we have to consider that user behavior patterns can have a change gradually. Thus, using long-term training sets can reduce prediction accuracy instead.

To demonstrate our inference, we have an analysis of most frequently used Apps. As shown in Figure 1, we count the used ratio of 8 Apps every 5 weeks. The number of tag such as 2 indicates the second five- weeks in the upper right corner. The 8 Apps are the most frequently used Apps for the user. The used ratio of every App varies with time. Thus, in term of the used frequency of Apps, we conclude the conclusion that user behavior pattern of using Apps varies with time.

Also, we demonstrate the inference from the view of the prediction accuracy. We explore the impact of the amount of training data ranged from one week to twelve weeks on the prediction accuracy. There are two schemes to get prediction accuracy from data set. The two schemes are shown in Figure 3. In Scheme 1, training periods and test periods alternately present and they have no overlap on the time dimension. From Figure 3(b), we can find that there exists overlapping parts on the time dimension in Scheme 2. And in Scheme 2, there is no time interval in two adjacent test periods. In both two schemes, after the i_{th} training period, we get prediction accuracy from the corresponding i_{th} test period based on the model trained in the i_{th} training period. In the Equation 2, ave_acc_user(the average value of all the average accuracy in every test period) represent the prediction accuracy on a user and acc_each is the prediction accuracy each test. n_1 is the number of test periods in dataset and n_2 is the times of test in each test period.

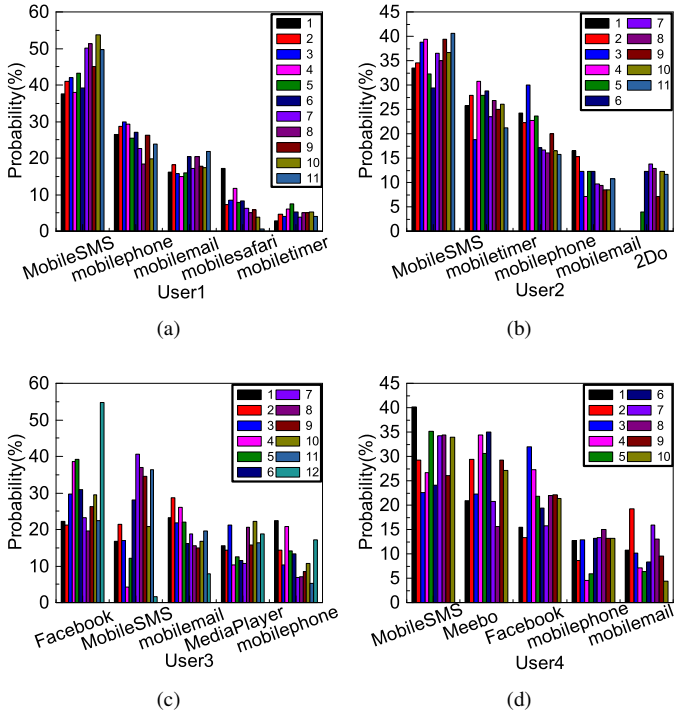

Figure 1: (a)(b)(c)(d)indicate that the used ratio of most frequently used 8 Apps on four different smartphone users.

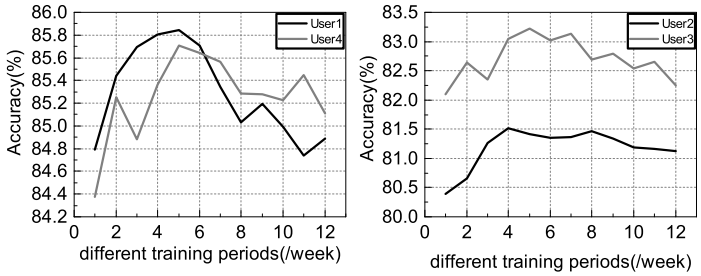

Figure 2: (a)(b)indicate that the changing trend of prediction accuracy along with increasement of the size of training sets.

$$ave\_acc\_user = \sum_{i=1}^{n_1} \sum_{j=1}^{n_2} acc\_each \quad (2)$$

In this paper, we adopt

As shown in Figure 2, the experimental results are based real traces collected from the 9 participants. And the k is set at 5. From the Figure 2, we can see that the prediction accuracy will rise at the beginning, but it will drop then. Namely, the accuracy is saturated at certain point from one week to twelve week. Therefore, the long-term training data does not necessarily achieve higher accuracy. Heuristically, it is because mobile users may install new Apps and have changed their behavior patterns(e.g. preferences) slightly. On

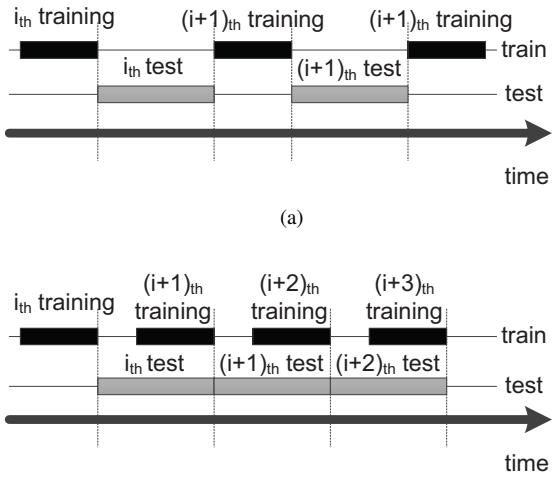

Figure 3: (a) (b) represents two different schemes to get prediction accuracy from data set. (a) describes Scheme 1 and (b) describe Scheme 2.

the other hand, more records(i.e., training data) more time required to make prediction.

We also conclude from Figure 2 that the positions the saturation points varies greatly on different mobile users. So it is necessary to have s adaptive scheme to adjust the amount of training data. However, it is difficult to select the optimal amount of training data directly. The work flow of App prediction is training-prediction-retraining-reprediction. We fixed the amount of training data and adjust the time span of the period of prediction instead. Because the precision accuracy is easily obtained in the period of prediction. We can adjust the time span based on the prediction accuracy.

From Figure **??**, we get two observations that provide useful insights into the design of our algorithm.

Observation 1: User behavior pattern of using Apps varies with time. And the recent used Apps are more useful to predict the next used Apps.

Observation 2: For different smartphone users, the optimal amounts of training data id different. Therefore, we propose an adaptive algorithm to solve the problem.

2.4 Adjusting the time span of the prediction period adaptively

From the above, choosing an approximate amount of training data can not only improve the prediction accuracy but also reduce the time required to train the prediction model. Because the prediction accuracy can be obtained in the period of prediction, we choose an a dynamic time span of prediction period instead of a dynamic amount of training data. We thus propose a strategies based on the minimization of the prediction error. It can be considered as *early stopping*.

Early stopping: In machine learning, early stopping is a form of regularization used to avoid over-fitting when training a learner with an iterative method, such as gradient descent. Such methods update the learner so as to make it better fit the training data with each iteration. Early stopping rules provide guidance as to how many iterations can be run before the learner begins to over-fit.

We have used the strategy of *early stopping* to achieve the adaptive time span of prediction periods. The detailed description is as follows. After training with a fixed amount of training data, the

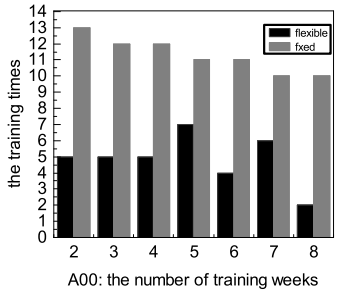

Figure 4: (a)(b)indicate that the changing trend of prediction accuracy along with increasement of the size of training sets.

achieved model by training is used to predict the Apps that will be used next. At this time, an approximate time span of prediction periods need to be considered. Too long time span of prediction periods can make the prediction accuracy decreased. But too short time span of prediction periods can increase the training costs(i.e., more training times are needed). In our strategy of *early stopping*, we first need to selected an initial time interval(e.g., two weeks). In the strategy, the shortest time span of prediction period is one week(i.e., the first week in the two weeks). The rest of the prediction time can be dynamically adjusted. Except from the first week, the average precision accuracy(acc_day) of the whole prediction period need to be calculated with every passing day. At the same time, the average prediction accuracy(acc_week) of the whole prediction period also need to be calculated with every passing week. But considering to improve the final prediction accuracy, the maximal acc_week(max_acc_week) is chosen as a baseline.

Intuitively, the acc_week is more stable than the acc_day, and the it is more representative than the acc_day. Thus, with every passing day, we calculate acc_day, max_acc_week and compare acc_day and max_acc_week. Provided the ratio of acc_day and acc_week is less than the threshold that we set, we have a time penalty for time span of prediction period(i.e., reducing the time span of prediction period). In the beginning, we have selected an initial time interval of two week. Excluding the first week, the second week is chosen as a baseline. When the prediction accuracy is too low, the time span of prediction period can be reduced on the basis of the baseline.

In the Algorithm2.1, the penalty function *penal_func* is the key. We have tried to use different formula. After a lot of experiments, we choose Formula4 as our final penalty function. In Formula4, $threshold$ is a number between 0 and 1 and a is a parameter. Heuristically, the rate of growth of the logarithmic function value is more and more gentle, which will effectively avoid certain accidental factors to reduce the time interval between two adjacent training.

$$ratio = \frac{\sum_{i=1}^{n} each\_acc}{n \times max\_ave\_acc} \tag{3}$$

$$penalty = 1 + \frac{log(threshold) - log(ratio)}{log(a)} \tag{4}$$

As shown in Figure4, our adaptive method nearly reduce half of the training times with flexible prediction period than using fixed prediction period(two weeks). In addition, in terms of the accuracy, the adaptive method nearly obtains the same prediction accuracy and even has an improvement for some users. The accuracy result of comparison is listed in Table 2.

Algorithm 2.1: EARLY STOPPING ALGORITHM

Input : L: an empty list; TS: a fixed time span; $Thres1$: a number between 0 and 1; $Thres2$: a number between 0 and 1;

Output : ave_acc_{total}: the average prediction accuracy in the whole prediction period;

$n\_predict \leftarrow 0$;
$sum\_accuracy \leftarrow 0$;
for *each* TS *in the firstweek of the prediction period* **do**
    Predict next used Apps and get the prediction accuracy P;
    $n\_predict \leftarrow n\_predict$+1;
    $sum\_accuracy \leftarrow sum\_accuracy$+$P$;
    if Sum *is an integer multiple of one day* **then**
        Add the tuple ($n\_predict$,$sum\_precision$) into list L;
        $n\_predict \leftarrow 0$;
        $sum\_accuracy \leftarrow 0$;
RS: the rest of prediction period; $RS \leftarrow$ the time of one week;
 $n\_predict \leftarrow 0$;
$sum\_accuracy \leftarrow 0$;
$flag$: a flag on whether to increase the time of prediction period;
$flag \leftarrow$ *True* **while** $flag$ **do**
    for *each* TS *in* RS **do**
        Predict next used Apps and get the prediction accuracy P;
        $n\_predict \leftarrow n\_predict$+1;
        $sum\_accuracy \leftarrow sum\_accuracy$+$P$;
        if Sum *is an integer multiple of one day* **then**
            Add the tuple ($n\_predict$,$sum\_precision$) into list L;
            $n\_predict \leftarrow 0$;
            $sum\_accuracy \leftarrow 0$;
            Calculate average accuracy ave_acc according to L;
            Get the maximal accuracy max_acc in the first one,two,... of prediction period **if** ave_acc ¡ $max_acc \times Thres1$ **then**
                Have a penalty for ave_acc(lowering the value);
    Calculate average accuracy ave_acc according to L;
    Get the maximal accuracy max_acc in the first one,two of prediction period;
    if ave_acc ¿ $max\_acc \times Thres2$ **then**
        Have an award for RS (add the value);
    $flag \leftarrow$ *False*;
Calculate ave_acc_{total} according to L;
return ave_acc_{total};

Table 2: the result of accuracy comparison between our adaptive method and adopting fixing prediction period method.

User	Strategy	training weeks(/week)							avg
		2	3	4	5	6	7	8	
A00	flexible(%)	84.86	84.30	84.57	84.77	84.57	84.77	84.03	84.55
	fixed(%)	83.89	84.48	83.78	84.25	84.21	84.04	84.11	84.11
A03	flexible(%)	89.58	89.51	89.58	89.53	89.51	89.54	89.64	89.55
	fixed(%)	89.51	89.22	89.07	89.38	89.53	89.59	89.64	89.42

A. Appendix Title

This is the text of the appendix, if you need one.

Acknowledgments

Acknowledgments, if needed.

References

J. Brändle, C. Eppler, and S. Möbius. Face recognition with deep learning for mobile applications. 2015.

B. Catanzaro. Deep learning with cots hpc systems. 2013.

W. Chen, J. T. Wilson, S. Tyree, K. Q. Weinberger, and Y. Chen. Compressing neural networks with the hashing trick. *CoRR, abs/1504.04788*, 2015.

Y. Cheng, X. Y. Felix, R. S. Feris, S. Kumar, A. Choudhary, and S.-F. Chang. Fast neural networks with circulant projections. *arXiv preprint arXiv:1502.03436*, 2015.

B.-G. Chun and P. Maniatis. Augmented smartphone applications through clone cloud execution. In *HotOS*, volume 9, pages 8–11, 2009.

M. Courbariaux, J.-P. David, and Y. Bengio. Training deep neural networks with low precision multiplications. *arXiv preprint arXiv:1412.7024*, 2014.

M. Courbariaux, Y. Bengio, and J.-P. David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

J. Dai, K. He, and J. Sun. Convolutional feature masking for joint object and stuff segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3992–4000, 2015.

M. Denil, B. Shakibi, L. Dinh, N. de Freitas, et al. Predicting parameters in deep learning. In *Advances in Neural Information Processing Systems*, pages 2148–2156, 2013.

E. L. Denton, W. Zaremba, J. Bruna, Y. LeCun, and R. Fergus. Exploiting linear structure within convolutional networks for efficient evaluation. In *Advances in Neural Information Processing Systems*, pages 1269–1277, 2014.

Y. Gong, L. Liu, M. Yang, and L. Bourdev. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*, 2014.

S. Han, H. Mao, and W. J. Dally. Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding. *CoRR, abs/1510.00149*, 2, 2015a.

S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015b.

B. Hariharan, P. Arbeláez, R. Girshick, and J. Malik. Hypercolumns for object segmentation and fine-grained localization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 447–456, 2015.

S. Harris. Your samsung smarttv is spying on you, basically. *The Daily Beast*, 2015.

J. Hauswald, Y. Kang, M. A. Laurenzano, Q. Chen, C. Li, T. Mudge, R. G. Dreslinski, J. Mars, and L. Tang. Djinn and tonic: Dnn as a service and its implications for future warehouse scale computers. In *ACM SIGARCH Computer Architecture News*, volume 43, pages 27–40. ACM, 2015.

G. Hinton, L. Deng, D. Yu, G. E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T. N. Sainath, et al. Deep neural networks for acoustic modeling in speech recognition: The shared views of four research groups. *IEEE Signal Processing Magazine*, 29(6):82–97, 2012.

A. Karpathy and L. Fei-Fei. Deep visual-semantic alignments for generating image descriptions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3128–3137, 2015.

A. Kosner. Client vs. server architecture: Why google voice search is also much faster than siri@ online, october 2012. *URL http://tinyurl. com/c2d2otr*.

A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

S. S. Latifi Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi. Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 1201–1205. ACM, 2016.

Q. V. Le. Building high-level features using large scale unsupervised learning. In *2013 IEEE international conference on acoustics, speech and signal processing*, pages 8595–8598. IEEE, 2013.

J. Long, E. Shelhamer, and T. Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3431–3440, 2015.

G. Mittal, K. B. Yagnik, M. Garg, and N. C. Krishnan. Spotgarbage: smartphone app to detect garbage using deep learning. In *Proceedings of the 2016 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pages 940–945. ACM, 2016.

A. Novikov, D. Podoprikhin, A. Osokin, and D. P. Vetrov. Tensorizing neural networks. In *Advances in Neural Information Processing Systems*, pages 442–450, 2015.

S. S. L. Oskouei, H. Golestani, M. Kachuee, M. Hashemi, H. Mohammadzade, and S. Ghiasi. Gpu-based acceleration of deep convolutional neural networks on mobile platforms. *arXiv preprint arXiv:1511.07376*, 2015.

P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *CoRR,abs/1409*, 2014.

O. Vinyals, A. Toshev, S. Bengio, and D. Erhan. Show and tell: A neural image caption generator. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 3156–3164, 2015.

K. Yanai, R. Tanno, and K. Okamoto. Efficient mobile implementation of a cnn-based object recognition system. In *Proceedings of the 2016 ACM on Multimedia Conference*, pages 362–366. ACM, 2016.

Q. Zhang. The convolutional neural network(cnn) for android. https://github.com/zhangqianhui/CnnForAndroid/. Accessed: 2016-05-30.