Enabling Privacy with Transfer Learning for Image Classification DNNs on Mobile Devices

Andreas Seiderer, Michael Dietz, Ilhan Aslan, Elisabeth André
University of Augsburg
86159 Augsburg, Germany
{andreas.seiderer,michael.dietz,ilhan.aslan,andre}@informatik.uni-augsburg.de

ABSTRACT

More people could benefit of Machine Learning (ML) as an increasingly important technology and service, if state-of-the-art ML techniques with training capability were accessible on personal devices. To this end, we report details on how to deploy TensorFlow on off-the-shelf mobile and embedded devices and retrain current deep neural networks for image recognition on-device. Our motivation is to both grant privacy and allow users to efficiently personalize image classifiers for their own needs and purposes, and thus contribute towards turning ML into a "social good", which benefits the largest number of people in the greatest possible way.

CCS CONCEPTS

• **Human-centered computing → Mobile devices**; • **Computing methodologies → Transfer learning**;

KEYWORDS

transfer learning, image recognition, mobile devices, personalization, privacy, neuronal network

ACM Reference Format:
Andreas Seiderer, Michael Dietz, Ilhan Aslan, Elisabeth André. 2018. Enabling Privacy with Transfer Learning for Image Classification DNNs on Mobile Devices. In *International Conference on Smart Objects and Technologies for Social Good (Goodtechs '18), November 28–30, 2018, Bologna, Italy.* ACM, New York, NY, USA, 6 pages. https://doi.org/10.1145/3284869.3284893

1 INTRODUCTION

It is undeniable that personal mobile technology has empowered individuals from all walks of life, starting an era of mobile and data-driven innovation. In this era, Machine Learning (ML) is becoming an increasingly important and ubiquitous service with the potential to benefit the largest number of people. For example ML based image classification could assist users in classifying anything by simply taking pictures. Doing so, a person with food-sensibility could, for example take pictures of their food to have algorithms help recognize and automatically log their food consumption over time. This could be a helpful addition to food logging apps, like the mobile multi-device system presented in [17], that already include taking photos. Furthermore, ML models could be personalized by people adding new pictures. However, it seems, ML's potential impact as a "social good" is limited, since many people seem to fear how their personal data will be handled and don't use personal-data driven ML services. Especially, vulnerable groups of people, such as people with health issues who might benefit most have good reasons for concerns. For example with recorded food images it is already possible to get information about the location of users or even their religious conviction [13]. It seems, for ML to become a tool or service for "all" people, it is desirable to have solutions granting both privacy and personalization.

In this paper we study the feasibility of having personalized on-device image classification on smartphones and embedded devices. We focus on deep neural networks, since they are currently the best performing classifiers. But, they usually require (i) a vast amount of data and (ii) high performance systems for training/learning. We believe that both issues can be addressed applying the transfer learning technique, which allows retraining existing models to new domains (e.g., transfer a flower recognition model into a food recognition model).

We report a series of simulations showing that transfer learning as a technique is feasible with limited resources on mobile and embedded devices. We try to make our measurements as reproducible as possible. For this reason we provide all (adapted) scripts / file lists in a Github repository[1]. The data set we use as an example is the freely available flower data set[2] providing five classes with a reasonable data amount for our tests. Before we provide our results in detail we summarize related ML methods to grant privacy and enable personalization.

2 BACKGROUND

Cloud services are a popular method on mobile devices to offload energy and time-consuming tasks to much more powerful servers. For instance, Polonio et al. [15] proposed an automatic photo tagging application for sharing pictures on social media, which relies on external services to retrieve the tags.

A system for mobile food recognition using transfer learning was presented by Temdee et al. [19]. Nevertheless, the system was just tested with the quite resource consuming Inception v3 model which was retrained on an external system and finally deployed to the smartphone. This is similar to traditional approaches where a generalizable model is trained with offline data and is then deployed to the target device for execution (e.g., [9]).

Goodtechs '18, November 28–30, 2018, Bologna, Italy

ACM ISBN 978-1-4503-6581-9/18/11…$15.00
https://doi.org/10.1145/3284869.3284893

[1] https://github.com/hcmlab/tf-transferlearning-benchmarks
[2] http://download.tensorflow.org/example_images/flower_photos.tgz

There are also hybrid systems that can decide to use on-device or cloud computation for more efficient usage of deep neural networks [10]. However, once the data leaves the user's device, privacy concerns might arise. Sensitive information about a user's private life, habits and relations could be extracted or inferred from the collected data [8]. This specifically applies to images since they can include a variety of information such as the current time, location, context, surrounding environment or nearby people.

In order to address these issues, several techniques have been developed. For example, one solution is to reduce the data fidelity by adjusting the sensing granularity and recording intervals [8]. This can range from reducing the rate at which samples are collected to automatically blurring people's faces in images. Another approach involves encrypting the private data on the user's device before processing it on a server [6, 20]. Through that, the resources of cloud services can be used without having to disclose the unencrypted data. Nevertheless, it might be possible to break the encryption. Therefore, the safest approach would be to perform all necessary calculations directly on the mobile device. Since the required computational resources for training complex neural networks currently exceed the performance of mobile devices, this is no feasible option for now.

However, instead of completely training a new machine learning model from scratch, an alternative might be to use a pretrained model which can be adapted to the current task. If the classes never change this can be provided by doing incremental learning. For example, this method has been utilized with an online naive bayes classifier on accelerometer data features to spot drink gestures on a smartwatch [11]. Nevertheless, this method is no longer applicable as soon as classes change. An alternative is *transfer learning* which is often applied in situations where we want to solve a classification problem in one domain but only have a sufficient amount of training data in another domain [14]. The problem with sufficient training data is especially existing when using deep neural networks which for image classification tasks are currently state-of-the-art. Since modifying an already trained model requires fewer resources, it might be possible to perform this task on mobile devices in a reasonable amount of time.

In this work we therefore want to get an idea of what time durations we have to expect on a selection of quite recent mobile / embedded consumer devices, especially for the on-device training process. Additionally, further performance optimizations could also be applied to reduce the required duration [7].

3 ON-DEVICE RETRAINING / CLASSIFICATION

3.1 TensorFlow for Training Models on Android

For our tests we decided to use Google's TensorFlow framework for neural networks, which already includes official examples for image retraining [5]. Android is officially supported but there is limited support to use training functions on Android. For this reason we decided to use a workaround, which allows to run all of our tests (i.e., identical Python scripts using the same code base) also on smartphone hardware. Nearly all Android devices use an ARM based CPU and there are unofficial instructions on how to build a "complete" TensorFlow for Raspberry Pi 2 / 3. More recently, nightly builds for Raspberry Pi were started to be built that are also compatible with other ARMv7 devices in a Debian Linux environment. In the following steps, we used one of these Python "wheels". To resolve all dependencies on Android a full Linux is required, for which a chroot environment was utilized. More specifically, we used Linux Deploy [1] with Debian on rooted Android devices.

3.2 Performance Measurement of Model Retraining

To get an impression whether the training process on smartphones and embedded devices is currently reasonably fast for our purpose of on-device image retraining, we conducted performance tests with different devices and models. It should be mentioned that our performance tests can just provide hints about the execution speed a recent version of the "normal" TensorFlow provides on ARM devices with NEON acceleration. That is, we didn't put extra effort in optimization.

3.2.1 Software. We used a nightly build of TensorFlow for Raspberry Pi based on version 1.7 with Python 3 bindings[3]. On all of our systems Debian or Raspbian Stretch was installed as native or chroot environment. On smartphones we installed the most recent versions of LineageOS available for the specific device with root access.

To be able to create comparable benchmarks we decided to modify the official image retraining script (retrain.py) provided by Google [5] by adding logging functions to write time stamps after tasks, like the creation of "bottlenecks", to a file. We use the term "bottleneck" like in [5] for the last layer before the final output layer. At this layer the highest data abstraction or reduction of dimension is reached.

The script conducts following steps: 1. scan for pictures in a directory, 2. download a specified base model from TensorFlow Hub, 3. create / save "bottlenecks", 4. adapt the final layer of the base model for the new class count, 5. retrain / evaluate and 6. save the model. To reduce any biases (e.g., caused by background programs) we ran every test three times.

3.2.2 Devices. For our performance tests we used two Android smartphones and two embedded ARM devices. Technical details can be found in Table 1. The performance of the integrated SoCs, except for the Raspberry Pi 3 B, which is not using a smartphone SoC, should give an overview about the performance today's low-end and mid-range smartphones can deliver.

We chose the two Nexus smartphones as these are usually being considered as being especially developer friendly and therefore are widely spread. The Nexus 4 was first released in November 2012, the Nexus 6P in September 2015.

In addition to the two smartphones, we included two ARM embedded boards in our tests. The Raspberry Pi 3 B is a very popular low priced (ca. 35 USD) low power embedded computer. It was released in February 2016. A more powerful embedded board is the Odroid HC1 with a price of ca. 49 USD. A version of this SoC has also been used for the Samsung Galaxy S5 (SM-G900H) released in April 2014.

[3] git revision: 5d33c1e49178aedbb459da7ce58eca710102c06b

	Nexus 4	Nexus 6P	Raspberry Pi 3 B	Odroid HC1
SoC	Qualcomm Snapdragon S4 Pro	Qualcomm Snapdragon 810 v2.1	Broadcom BCM2837	Samsung Exynos5422
CPU	4 x 1.5 GHz Krait	4 x 2.00 GHz Cortex-A57 & 4 x 1.55 GHz Cortex-A53	4 x 1.2 GHz Cortex-A53	4 x 2.0 GHz Cortex-A15 & 4 x 1.4 GHz Cortex-A7
RAM	2 GB, LPDDR2	3 GB, LPDDR4	1 GB, LPDDR2	2 GB, LPDDR3
Storage	NAND, 16 GB	NAND, 64 GB	SD card	SD card (OS) + SSD (Data)
Host-OS	Android 7.1 (LineageOS 14.1)	Android 8 (LineageOS 15.1)	Raspbian Stretch	Ubuntu Server 16.04.4 LTS
CHROOT	Linux Deploy: Debian Stretch armhf	Linux Deploy: Debian Stretch armhf	-	Docker: Debian Stretch (arm32v7/debian:stretch)

Table 1: Used ARM devices including host OS and OS of the test environment

3.2.3 Models. For our tests we selected several well-known recent convolutional neural networks for image recognition that are included by TensorFlow Hub and are pretrained on ImageNet (ILSVRC-2012-CLS). These are *Inception v3* [18] and *MobileNet* v1 [12] and v2 [16]. All top 1 and top 5 accuracies of the selected models can be found in Figure 2 and were collected from [2–4]. The size in percent of the MobileNets is the second to last value while the picture size in pixels is the last value of the name. Details on the models can be found in the according papers.

For high recognition rates we include *Inception v3* and *MobileNet_v2_1.4_224* which can provide a top 1 accuracy of 78.0 % respectively 75.0 % and a top 5 accuracy of 93.9 % respectively 92.5 %. In addition, we included smaller models that still provide relatively high accuracies.

For mobile usage the *MobileNet* v1 and v2 neural networks with different input sizes and network sizes were designed. From *MobileNet v1* we chose the ones with 100 % and 75 % size and an input picture size of 224x224 (biggest) and 128x128 (smallest). From *MobileNet v2* we selected the nets with 140 %, 100 % and 75 % size with the input picture sizes of 224x224 (biggest), 128x128 (smallest comparable with *MobileNet v1*) and 96x96 (smallest).

3.2.4 Data set. For the performance evaluation we used the flower data set from the TensorFlow tutorial. It consists out of 3670 jpg files that contain pictures of flowers of five classes (daisy: 633 files, dandelion: 898, roses: 641, sunflowers: 699, tulips: 799). The average size of the photos is about 62.0 kiB.

For serious accuracy comparisons between the selected models this data set is too small as all accuracies of the models, using the default split between training, validation and test set of the retrain.py script, lie between 86.4 % and 91.2 % for the test set. Figure 2 should provide a better idea about their general classification performance.

3.3 Performance and Resources

During our performance tests, all devices were connected to an external power supply. For the tests we used the adapted retrain.py script as described before. In Figure 1 the results of our tests are visible. The runtimes in seconds of the bottleneck creation, training and evaluation for each model are shown. The model names are additionally abbreviated: "Inc. v3" stands for *Inception v3*, the "M1" for *MobileNet v1* and "M2" for *MobileNet v2*.

The results clearly show that the bottleneck creation takes most of the time although it uses all available cores on all devices. On the Nexus 6P just the four "big" cores of the big.LITTLE SoC are used. On the Odroid HC1 all cores are used so that it was always the fastest device of our tests. For the training process just one core was used. Nevertheless, as expected the training consumed several times less time than the bottleneck creation. For the training process just the already calculated bottleneck files were used to train one layer. The Nexus 4 and Raspberry Pi 3 B usually require quite similar time amounts. Although the Pi 3 uses a more recent ARM architecture, it is visible that the training always has the biggest time share. This is probably a result of its lower CPU clock in comparison to all other devices, which is especially apparent on single core tasks like the training. The usually slower SD card could also introduce additional slowdowns.

In general, it is visible that the *Inception v3* model on all devices requires at least two times more time for the bottleneck creation in comparison to all tested "MobileNets" due to its higher complexity. The "M1_100_128", "M1_075_128" and "M2_100_096" show comparable low runtimes on all devices. Our fastest smartphone, the Nexus 6P, requires about 15 minutes for bottleneck creation for these models and about 4 minutes and 30 seconds to retrain the models. This is much faster than with the Inception v3 model which takes about 127 minutes (ca. 28.3 times slower) to create the bottlenecks and about 8 minutes (ca. 1.8 times slower) to retrain the model on this device.

The file size for all bottleneck files, which are stored as uncompressed text files, varies between 77.3 MiB (*Inception v3*) and 21.7 MiB (M1_075_128), which are easy to save and could still be heavily compressed in comparison to the original jpg files' data amount of 222.1 MiB. The resulting retrained models, saved as protocol buffer files, have a file size of about 83.4 MiB (Inception v3) down to 5.4 MiB ("M2_075_128" / "M2_075_224"), which is also no problem for mobile devices.

3.4 Recognition with Increased Image Amounts

To be able to observe the model quality during usage we simulated in this test an increasing number of pictures for several classes. For our tests we always retrained from the base model. We used the "M2_100_128" as it showed good recognition accuracies according

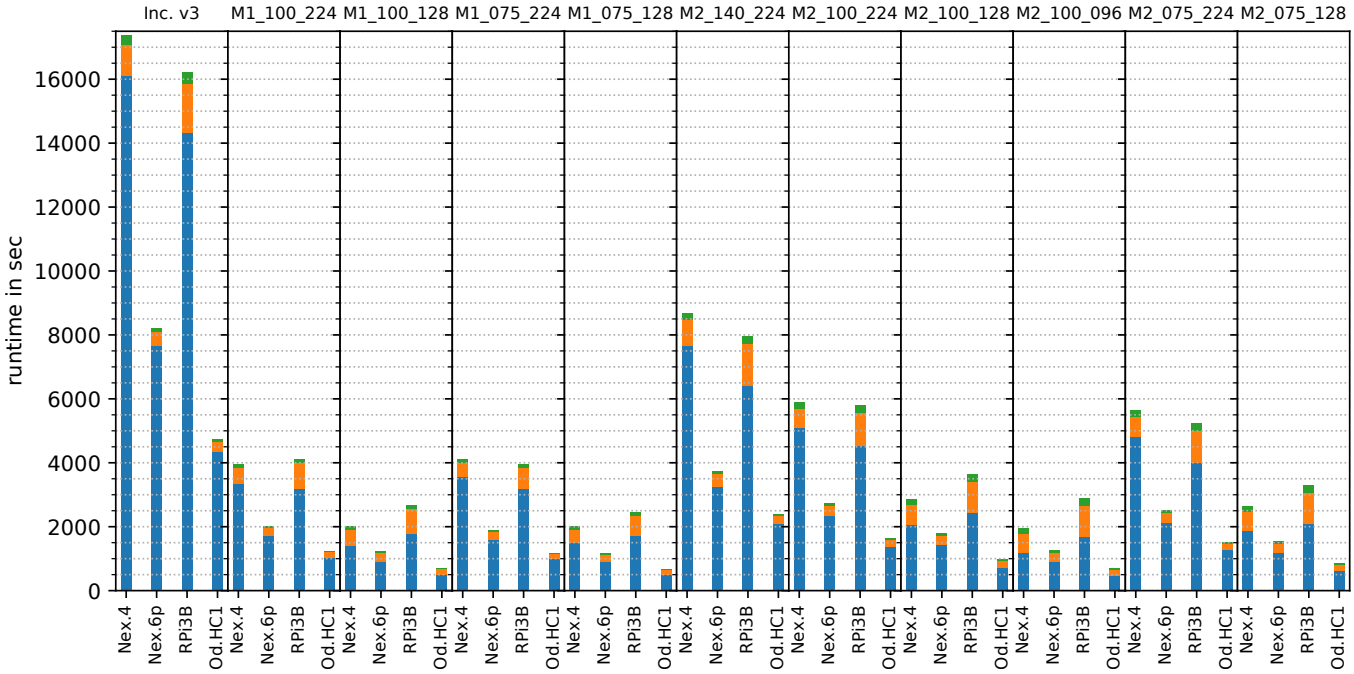

Figure 1: Average device runtimes for bottleneck creation (blue), training (orange) and evaluation (green) out of three runs for each model configuration for the flower data set.

Model name	Top 1	Top 5
Inception_v3_299	78.0 %	93.9 %
MobileNet_v1_1.0_224	70.9 %	89.9 %
MobileNet_v1_1.0_128	65.2 %	85.8 %
MobileNet_v1_0.75_224	68.4 %	88.2 %
MobileNet_v1_0.75_128	62.1 %	83.9 %
MobileNet_v2_1.4_224	75.0 %	92.5 %
MobileNet_v2_1.0_224	71.8 %	91.0 %
MobileNet_v2_1.0_128	65.3 %	86.9 %
MobileNet_v2_1.0_96	60.3 %	83.2 %
MobileNet_v2_0.75_224	69.8 %	89.6 %
MobileNet_v2_0.75_128	63.2 %	85.3 %

Figure 2: Accuracies for ILSVRC-2012-CLS data set.

to Table 2 but still had a comparably low runtime for bottleneck creation and training on all devices.

We did the retraining with the flower data set but used an adapted version of the "retrain.py" script allowing us to control, which pictures were used for training, validation and testing. For all tests, we used 400 pictures of each class for the model evaluation in the test-set, which were not used in the training or validation set. For the training and validation set a split of 80 % / 20 % was used for newly added files. Except for the training step count being set to 1000, the options of the script were kept the same as in the previous test. All tests were run three times on a desktop PC with TensorFlow 1.8 and are averaged.

3.4.1 Imbalanced: Incrementing one class. For our first test we were simulating a user constantly adding five new pictures to a new class. In this case the class is "daisy". The count of five pictures seems to be a reasonable amount of pictures a user is willing to make of an object he or she wants to detect later on. We began with a four class model that did not contain the "daisy" class but all other four flower classes. The other classes constantly consisted out of 150 pictures (training and validation set) per class. We started with adding 20 daisy pictures and incremented the picture count by five until 150 pictures were reached.

The model evaluation results are visible in Figure 3. It displays the macro-averaged precision and recall of each model evaluation with 400 pictures for each class. In the beginning it is visible that after adding the first 20 daisy pictures the precision but mainly the recall drop from about 86 % to about 84 % / 83 % as a new class is introduced. From that point on especially the recall increases with additional daisy pictures. The difference between precision and recall is getting lower as the picture counts per class are getting closer to each other.

3.4.2 Balanced: Incrementing all classes. For our second test we were simulating a user constantly adding five new pictures to all five flower classes. The model evaluation results are shown in Figure 4. It is visible that the difference between precision and recall is quite low and that both increase until 60 pictures per class are reached. Adding more pictures didn't have much effect on these scores.

3.4.3 Imbalanced: Randomly incrementing one class. For our third test we were simulating that a user is constantly adding five new pictures to one of the classes starting with zero pictures per class. If two classes contained pictures the training process could start, since for a classification task at least two classes are required.

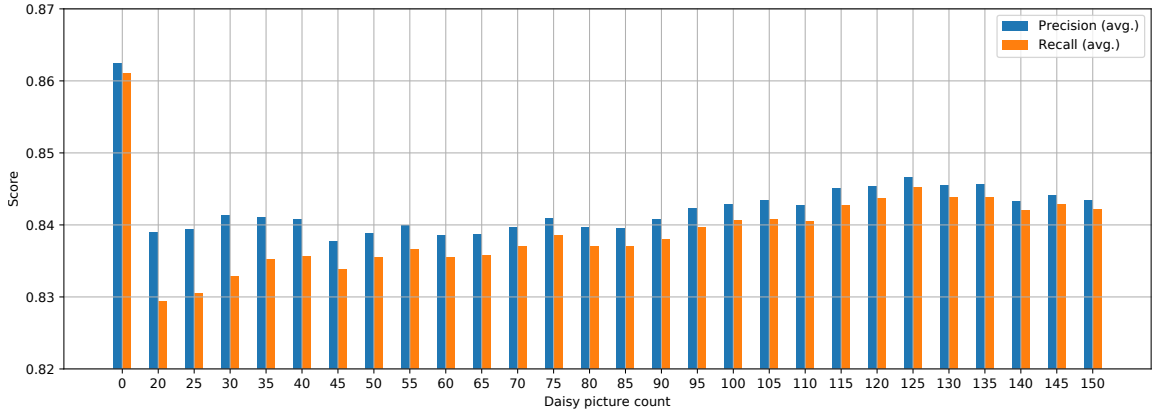

Figure 3: Macro-average precisions and recalls of models trained with an incremented count of daisy flower pictures while four other flower classes are kept with constant picture count of 150 (80 % training and 20 % validation). The evaluation set consists out of 400 pictures for each class. All trainings and model evaluations were conducted three times and the macro-averages of precisions and recalls are averaged.

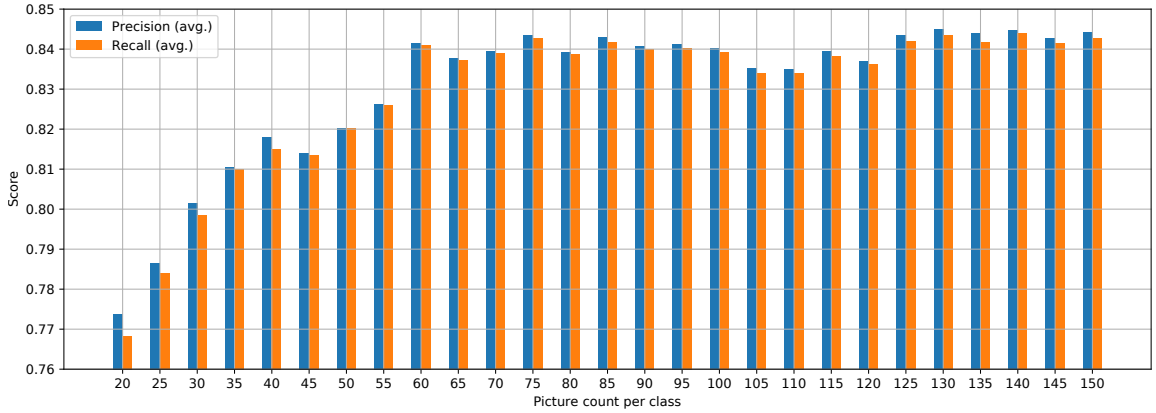

Figure 4: Macro-average precisions and recalls of models trained with an incremented count of flower pictures (80 % training and 20 % validation). The evaluation set consists out of 400 pictures for each class. All trainings and model evaluations were conducted three times and the macro-averages of precisions and recalls are averaged.

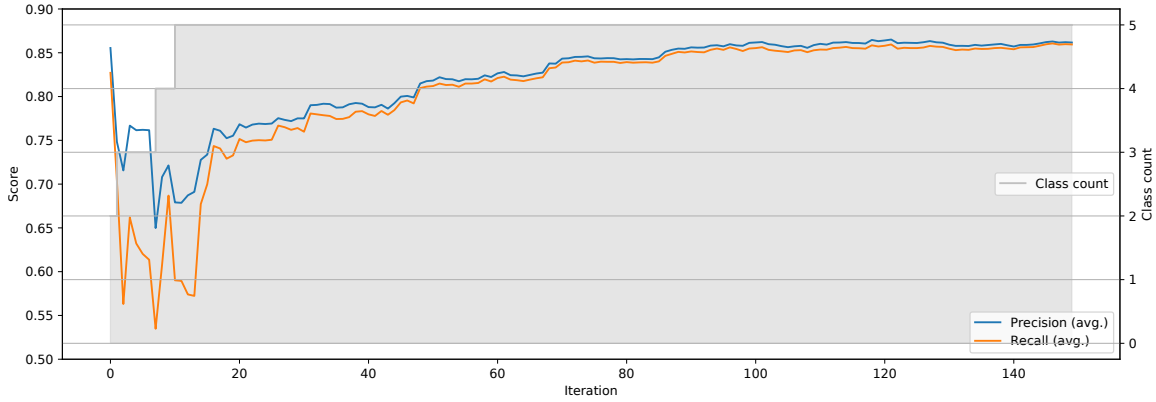

Figure 5: Macro-average precisions and recalls of models trained with an incremented count of flower pictures (80 % training and 20 % validation). The count of classes is shown in gray. New pictures are added to a randomly selected class until all classes consist out of 150 pictures. The evaluation set consists out of 400 pictures for each class. All trainings and model evaluations were conducted three times (stable random class selection) and the macro-averages of precisions and recalls are averaged.

The results of the test are visible in Figure 5. At the beginning, when the class count increases the precision and recall scores are quite unstable. Starting at iteration 10 a total of 50 (10*5) pictures are distributed between all five classes. With increasing number of iterations (and pictures) the precision and recall increase. The difference between precision and recall decreases since the classes are getting less imbalanced.

4 DISCUSSION AND CONCLUSION

We have presented performance tests with smartphones and embedded systems and reported results providing insights on feasibility of doing on-device transfer learning, considering required system resources and time.

After that, we performed three simulations to see how incrementally added new pictures can be included in the retraining process to create new usable models. In the first simulation we assumed that a user is adding data of just one new class to a hypothetical system resulting in a very imbalanced data set for the retraining. The second simulation assumed that a model is retrained with an over all classes equally distributed data set. The last simulation gave hints about the model qualities to be expected when a user randomly adds pictures to different classes.

The performance measurements show that image retraining on mobile and embedded devices with neural networks is possible in a reasonable time with a data set of 3760 pictures even without additional hardware acceleration (DSP or GPU). Especially the smaller models optimized for mobile usage showed acceptable performance and storage consumption so that it would be possible to personalize an existing model with user-provided images on the device. The retraining process could be triggered during phases where the device is not used - a smartphone for example during charging. The tests with an increasing picture amount show that already small image counts can provide reasonably good recognition accuracies even if they are spread imbalanced over the classes. If new classes are introduced it is clear that the recognition rates drop and more pictures have to be added until especially the recall stabilizes again.

The implications of these results can be illustrated, for example with the privacy sensitive use case of a nutrition logging application on a smartphone. Especially self-made dishes can introduce issues with generic classifiers since they can look very different, making it reasonable to personalize the model. Since users of nutrition logging apps often already take pictures of every meal and then manually add labels, it would be convenient to use these images also for training purposes. Thus, a first retrained classifier model could be created quickly after adding a relatively small amount of pictures (around 5) to at least two classes. At this point the model could already be used, e.g. to automatically label new nutrition pictures. Adding more pictures to the classes would further improve the overall recognition performance of the model until the capabilities of the neural network structure are reached.

We believe, in order to improve the practical usage of personalized models, future work could explore (i) automatic and manual mechanisms to undo eventually broken models (e.g. after adding poorly taken photos to a class), (ii) users' experience of creating their own personalized models and (iii) further techniques to reduce resource consumption for the training process e.g. by combining transfer learning with incremental learning whenever the classes didn't change. We hope the insights we have presented will also inspire fellow researchers, and ML on mobile devices will soon be considered as a "social good" accessible for all.

REFERENCES

[1] 2018. Linux Deploy. https://play.google.com/store/apps/details?id=ru.meefik.linuxdeploy accessed April 6 2018.

[2] 2018. Mobilenet v1 accuracies. https://github.com/tensorflow/models/blob/master/research/slim/nets/mobilenet_v1.md accessed April 6 2018.

[3] 2018. Mobilenet v2 accuracies. https://github.com/tensorflow/models/tree/master/research/slim/nets/mobilenet accessed April 6 2018.

[4] 2018. Model accuracies. https://github.com/tensorflow/models/blob/master/research/slim/README.md accessed April 6 2018.

[5] 2018. TensorFlow retraining. https://www.tensorflow.org/tutorials/image_retraining accessed April 6 2018.

[6] Hervé Chabanne, Amaury de Wargny, Jonathan Milgram, Constance Morel, and Emmanuel Prouff. 2017. Privacy-preserving classification on deep neural network. *IACR Cryptology ePrint Archive* 2017 (2017), 35.

[7] C. J. Chen, K. C. Chen, and M. c. Martin-Kuo. 2018. Acceleration of neural network model execution on embedded systems. In *2018 International Symposium on VLSI Design, Automation and Test (VLSI-DAT)*. 1–3. https://doi.org/10.1109/VLSI-DAT.2018.8373246

[8] Delphine Christin, Andreas Reinhardt, Salil S Kanhere, and Matthias Hollick. 2011. A survey on privacy in mobile participatory sensing applications. *Journal of systems and software* 84, 11 (2011), 1928–1946.

[9] Michael Dietz, Daniel Schork, Ionut Damian, Anika Steinert, Marten Haesner, and Elisabeth André. 2017. Automatic Detection of Visual Search for the Elderly using Eye and Head Tracking Data. *KI - Künstliche Intelligenz* 31, 4 (01 Nov 2017), 339–348. https://doi.org/10.1007/s13218-017-0502-z

[10] Amir Erfan Eshratifar and Massoud Pedram. 2018. Energy and Performance Efficient Computation Offloading for Deep Neural Networks in a Mobile Cloud Computing Environment. In *Proceedings of the 2018 on Great Lakes Symposium on VLSI (GLSVLSI '18)*. ACM, New York, NY, USA, 111–116. https://doi.org/10.1145/3194554.3194565

[11] Simon Flutura, Andreas Seiderer, Ilhan Aslan, Chi-Tai Dang, Raphael Schwarz, Dominik Schiller, and Elisabeth André. 2018. DrinkWatch: A Mobile Wellbeing Application Based on Interactive and Cooperative Machine Learning. In *Proceedings of the 2018 International Conference on Digital Health (DH '18)*. ACM, New York, NY, USA, 65–74. https://doi.org/10.1145/3194658.3194666

[12] Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. *CoRR* abs/1704.04861 (2017). arXiv:1704.04861 http://arxiv.org/abs/1704.04861

[13] Marcus Maringer, Pieter van't Veer, Naomi Klepacz, Muriel C. D. Verain, Anne Normann, Suzanne Ekman, Lada Timotijevic, Monique M. Raats, and Anouk Geelen. 2018. User-documented food consumption data from publicly available apps: an analysis of opportunities and challenges for nutrition research. *Nutrition Journal* 17, 1 (09 Jun 2018), 59. https://doi.org/10.1186/s12937-018-0366-6

[14] S. J. Pan and Q. Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (Oct 2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191

[15] Davide Polonio, Federico Tavella, Marco Zanella, and Armir Bujari. 2018. GHio-Ca: An Android Application for Automatic Image Classification. In *Smart Objects and Technologies for Social Good*, Barbara Guidi, Laura Ricci, Carlos Calafate, Ombretta Gaggi, and Johann Marquez-Barja (Eds.). Springer International Publishing, Cham, 248–257.

[16] Mark Sandler, Andrew G. Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. 2018. Inverted Residuals and Linear Bottlenecks: Mobile Networks for Classification, Detection and Segmentation. *CoRR* abs/1801.04381 (2018). arXiv:1801.04381 http://arxiv.org/abs/1801.04381

[17] Andreas Seiderer, Simon Flutura, and Elisabeth André. 2017. Development of a Mobile Multi-device Nutrition Logger. In *Proceedings of the 2Nd ACM SIGCHI International Workshop on Multisensory Approaches to Human-Food Interaction (MHFI 2017)*. ACM, New York, NY, USA, 5–12. https://doi.org/10.1145/3141788.3141790

[18] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jonathon Shlens, and Zbigniew Wojna. 2015. Rethinking the Inception Architecture for Computer Vision. *CoRR* abs/1512.00567 (2015). arXiv:1512.00567 http://arxiv.org/abs/1512.00567

[19] Punnarumol Temdee and Surapong Uttama. 2017. Food recognition on smartphone using transfer learning of convolution neural network. In *Wireless Summit (GWS), 2017 Global*. IEEE, 132–135.

[20] Q. Zhang, L. T. Yang, and Z. Chen. 2016. Privacy Preserving Deep Computation Model on Cloud for Big Data Feature Learning. *IEEE Trans. Comput.* 65, 5 (May 2016), 1351–1362. https://doi.org/10.1109/TC.2015.2470255