Experiences Developing a Computer Vision SDK for Mobile Apps

Julian Harty*
The Open University
Milton Keynes, UK
julianharty@gmail.com

ABSTRACT

We developed a mobile SDK that ran several ML models on device that was used by app developers who integrated the SDK into their app to provide real-time guidance and decisions based on what the on-device camera and other inputs provided. The SDK was instrumented with mobile analytics. We encountered various challenges on the journey and at least some of these might be of research interest.

CCS CONCEPTS

• **Computing methodologies** → **Neural networks**; Supervised learning; • **Software and its engineering** → *Software design trade-offs*; • **Applied computing** → *E-commerce infrastructure*; • **Computer systems organization** → *Real-time system architecture.*

KEYWORDS

Empirical Software Engineering, Machine Learning, Mobile Analytics, Mobile Apps, Mobile Software Development, SDK, Software Monitoring

ACM Reference Format:
Julian Harty. 2024. Experiences Developing a Computer Vision SDK for Mobile Apps. In *IEEE/ACM 9th International Conference on Mobile Software Engineering and Systems (MOBILESoft '24), April 14–15, 2024, Lisbon, Portugal.* ACM, New York, NY, USA, 3 pages. https://doi.org/10.1145/3647632.3651393

1 INTRODUCTION

This paper is an extended abstract to posit various practical challenges that may also be of interest to the research community as the research may also lead to adoption of the results of that research across the industry. There are many SDKs (Software Development Kits) for mobile app developers, however only a small proportion include machine learning models running within the SDK, and similarly a small proportion incorporate mobile analytics to assess the behaviours of the SDK.

*ORCID 0000-0003-4052-0054

MOBILESoft '24, April 14–15, 2024, Lisbon, Portugal

ACM ISBN 979-8-4007-0594-6/24/04
https://doi.org/10.1145/3647632.3651393

Some details have been elided, modified, or perturbated to protect commercially sensitive material.

The business developed software and services that incorporate computer vision to process images taken on mobile devices including widely available smartphones running Android or iOS. At the start of this example third-party mobile apps took photos of specific business-related events such as delivery of a parcel, parking a rented vehicle, and so on. These third-party apps sent these photos to the business to be analysed immediately, they received a business decision within a few seconds. The photos were either sent directly by the app or via server software used by our clients; in both cases they called APIs that the business provides. The analysis was performed using various machine learning models running on CPUs and GPUs in the Cloud combined with additional data such as the location the photo was taken.

End to end processing including returning the results took several seconds - fast compared to humans processing the images at scale, but not fast enough to provide real time guidance and feedback to the people who took the photos. Furthermore, many photos were flawed in terms of their contents, *e.g.* of someone's foot rather than the desired subject *e.g.* a parcel or the rented vehicle. Asking users to take replacement photos seconds later could delay or disrupt their other actions and some clients chose not to ask rather than risk a poor user experience of their app or service.

We decided to create a cross-platform SDK for mobile apps where the SDK would include one or more on device compiled machine learning models with the aim of reducing latency sufficiently to provide real time live feedback to the person taking the photo with three goals:

(1) Improving the contents of the photos by guiding the end user to include key content in the photo that's used by the machine learning model(s) to make the decision.
(2) Improving the actions of end users, for example to park well, to deliver the parcel in good condition to the appropriate destination.
(3) Performing the immediate processing on the end user's smartphone, rather than having to rely on good network connectivity.

These goals were iterative in nature; we did not know *a priori* how much we could achieve, nor how much improvements would be 'good enough'.

2 REAL WORLD REQUIREMENTS

Some of these requirements emerged during the project, for example to remove a clash in the choice of mobile analytics library.

The overall SDK needed to have a significantly smaller footprint than the app in three areas in particular: 1) size of the static binary

including any ML models packaged as part of the SDK, 2) network traffic, 3) runtime performance on device. As per the improvements absolute values were hard to ascertain, instead we chose to adopt and use relative comparisons.

2.1 Emergent requirements

2.1.1 Support for flutter. Support for the programming languages and frameworks used for the app codebase. One of the early client developed their apps in flutter which led to us developing various proofs-of-concept including a demonstration app written in flutter for our SDK.

2.1.2 Mobile Analytics in common. What happens when both the SDK and the app have chosen the same mobile analytics offerings? there are potential clashes particularly where in the resulting binary file a single instance of the mobile analytics SDK is integrated. The clashes include: a) which account will receive the mobile analytics information, b) bleeding of information in the mobile analytics reports where the SDK account may receive events reported by the app, and vice-versa, threading and shared data structures where the mobile analytics SDK was implemented to assume a single account would use it, rather than several concurrently.

2.1.3 SDK's retrospectively not supported. As noted in [7] Sentry, a mobile analytics provider, confirmed their SDK was not officially supported in SDKs [11]. This information was only made public several months into the project and long after Sentry had been selected and incorporated into the SDK. The more common use case for mobile analytics is for apps to use them, rather than for SDKs to use them, and even rarer is when both the app and the SDK wish to use the same mobile analytics SDK.

3 SOFTWARE DEVELOPMENT PRACTICES

This section provides a summary of pertinent aspects of the software development practices of the team.

Three people formed the core group of developers: a mobile developer worked full time who wrote and tested the Swift and flutter code, the ML models were developed part-time, and the author as CTO also worked part-time in all aspects of the development project. Another seven or so people also contributed to the development, including several ML engineers and several full-stack developers.

The ML models were trained using Kedro and Tensorflow then converted to CoreML using a utility provided by Apple for this purpose. Both formats of the resulting ML models (CoreML and Tensorflow) were evaluated using validation and test image-sets including analysis of confusion matrices. Performance of the CoreML models on device was measured using Instruments, provided in Xcode, the IDE for iOS development, using practices described by Apple in [1, 3].

A mix of iPhones owned by individual members of the team and owned by clients were used during the project to test and evaluate the performance and behaviour of the SDK including the iterations of the ML model development and fine-tuning. These devices included the iPhone models 7, SE 2022, X, 12, 13, and 14.

4 REAL WORLD CHALLENGES

4.1 Latency

The first attempt to create an on device ML model had mixed results. One of the production models, written in PyTorch, was converted to work with PyTorch's PyTorch Mobile framework. While the model did run, it took approximately 40 seconds to process the image and return a result. Additional complexities with PyTorch Mobile for iOS did not bode well so we abandoned this approach and switched to using Kedro and tensorflow as initial experiments indicated the latency was at least 10x lower *i.e.* better performance.

4.2 Remote observability

Several distinct aspects of remote observability emerged during the work: a) the performance of the ML model on device, b) the reliability and performance of the SDK including detecting and reporting errors at runtime, c) the overall user experience - particularly from the UI and UX aspects of what the camera 'saw' and what appeared on screen. As mentioned earlier, Sentry was the initial choice of mobile analytics and intended to capture the performance and reliability data, however it had to be replaced for a particular client and we replaced it with PostHog; coincidentally both Sentry and PostHog are commercial products with open-source codebases. The market leader, Firebase, was rejected as unsuitable as it is designed to collect more user-oriented data than we deemed appropriate.

We implemented dual recording of the camera inputs, as frames, and as a video stream. The video stream was only enabled for internal builds of the SDK to protect the privacy of end users and to cap network traffic to volumes considered acceptable for end users. Again for pre-release testing purposes of the on-device ML models the SDK could submit all the recorded frames with the decision outcome that the on-device model predicted. These were then compared to the results of processing each of these frames with the cloud-hosted (larger) ML models to consider differences in their respective predictions.

4.3 Operational costs

After the first commercial client had integrated the SDK into their mobile app they configured the app to only actively use the SDK for whitelisted user accounts for field testing, *i.e.* less than 20 people globally of the 20,000 active userbase. Roughly 10 days into this period, one of the customer support team of the analytics service (PostHog) sent an email to ask whether the projected costs for the month of several thousand pounds was expected - it wasn't!

On investigation, their iOS SDK defaulted to recording Lifecycle events and this code was activated when the app was intialised [10], therefore the Lifecycle analytics events were being captured for the entire active userbase rather than only for the whitelisted users who were served by the SDK. The default setting was replaced by explicitly disabling the collection of Lifecycle events and an updated version of the SDK was provided to the client who incorporated it into their app and deployed a new release via the Apple App Store to their userbase. This significantly reduced the volume of analytics data and therefore the costs also reduced proportionally. The set of analytics events generated by the SDK were also

reviewed and revised as part of the improvements to balance the tradeoff between operational costs (for the analytics service provided by PostHog) and the ability to detect potential problems and bugs in operational use of the SDK.

4.4 Remote updates of models and decision maps, etc.

App updates in the Apple and Google Android ecosystems are staged and dependent on Apple's App Store and Google Play respectively. SDKs used by apps are compiled into the app's binary and released when the app is released. Updates to newer versions of an SDK are performed by the app developers (explicitly or implicitly) during development activities and before the app is released. For practical reasons the models and the decision maps using in the SDK shouldn't be dependent on new releases of apps being deployed onto end user devices (which can take days or weeks), for example, if the parking rules change for a city on a particular date, the decision maps on device need to be updated accordingly and in time for the change.

Apple CoreML documentation documents an approach to install and provide updates to models over a network connection [4]. We encountered multiple issues with this approach, and in short, it no longer appears to a) work, or b) be supported by Apple. We then chose to implement a very basic mechanism to update the ML models via our servers as an interim approach with the aim of finding or developing a more capable mechanism as soon as practical.

4.5 Nothing captured on some devices

The SDK did not receive any frames for one person at an early client. No error was reported by the SDK, and when the individual kindly provided their Apple smartphone, no errors were reported in the IDE - Xcode. The issue was reported to Apple's Developer Support who responded but without providing a technically valid response. The same issue is documented on StackOverflow [5] and as yet there is no known cause for the lack of frames. The eventual workaround was for the SDK to detect the no frames and then signal the app so it could revert to the app capturing the photo and then using the API to analyse the resulting image.

5 RESULTS

In six months the team were able to create, improve, and launch a mobile SDK that collected and processed 30 frames per second (FPS) of images with an on device ML model which performed (from a computing perspective) acceptably on all and any of Apple's iOS smartphones ranging from the iPhone 7 to the then latest iPhone 14 Pro Max. The processing time per frame ranged from approximately 30 mS on the iPhone 7 to 1.4 mS on the iPhone Pro Max (which performed most of the image processing using the GPU).

From a business perspective, the early client has been happy with the improvements in user experience and in the results of using the SDK rather than their previous approach.

6 RELATED WORK

The use of mobile analytics for logging purposes in opensource Android apps was discussed in [8], this work extends that work in the use of mobile analytics for closed source Android and iOS SDKs, two topics not covered previously. The use of mobile analytics to improve the reliability in mobile apps is discussed in depth in my PhD thesis [6]; this work builds on that research in the design, implementation, and operational aspects of using mobile analytics for an SDK and to measure on-device ML models in use.

A brief comparison of the performance of three ML model architectures on iOS devices assesses latency and frames per second on a range of then current iOS devices [12]. CoreML is documented online in Apple's developer documentation [2]. A more academical assessment is provided in [9]; from a practical perspective the book is out of date given the rapidity of changes to CoreML and related services provided by Apple.

7 FUTURE WORK

One of the key practical challenges is the ability to manage, track, and update ML models and decision maps in mobile apps and/or mobile-oriented SDKs.

ACKNOWLEDGMENTS

To the CEO and the engineering teams involved the underlying work, they know who they are. The customer success representative at PostHog who notified the team of the impending charges and who waived the unexpected charges when the SDK was deployed by the first external client.

REFERENCES

[1] Apple Developer. 2022. *Optimize your Core ML usage.* Apple Inc. https://developer.apple.com/videos/play/wwdc2022/10027/
[2] Apple Developer. 2024. *Core ML - Apple Developer Documentation.* Apple Inc. https://developer.apple.com/documentation/coreml
[3] Apple Developer. 2024. *Core ML Overview - Machine Learning - Apple Developer.* Apple Inc. https://developer.apple.com/machine-learning/core-ml/
[4] Apple Developer. 2024. *Downloading and Compiling a Model on the User's Device.* Apple Inc. https://developer.apple.com/documentation/coreml/downloading_and_compiling_a_model_on_the_user_s_device
[5] Archid. 2023. https://stackoverflow.com/questions/77287414/captureoutput-didoutputfrom-delegate-method-not-called-on-one-particular-ip
[6] Julian Harty. 2023. *Improving Application Quality using Mobile Analytics.* Ph. D. Dissertation. The Open University. https://oro.open.ac.uk/90629/
[7] Julian Harty, Ivan Dlugos, and Giancarlo Buenaflow. 2023. https://github.com/getsentry/sentry-dart/issues/1618
[8] Julian Harty, Haonan Zhang, Lili Wei, Luca Pascarella, Maurício Aniche, and Weiyi Shang. 2021. Logging Practices with Mobile Analytics: An Empirical Study on Firebase. In *2021 IEEE/ACM 8th International Conference on Mobile Software Engineering and Systems (MobileSoft).* IEEE, Madrid, Spain, 56–60. https://doi.org/10.1109/MobileSoft52590.2021.00013
[9] Oge Marques. 2020. *Machine Learning with Core ML.* Springer International Publishing, Cham, 29–40. https://doi.org/10.1007/978-3-030-54032-6_4
[10] PostHog. 2024. iOS - Docs - PostHog. https://posthog.com/docs/libraries/ios
[11] Sentry. 2023. https://docs.sentry.io/platforms/
[12] Anirudh Tulasi, S Malarselvi, and Mansa Devi Pappu. 2021. A Review on MobileNet, ResNet and SqueezeNet for iOS & iPadOS for on Device Training and Prediction using CoreML. *International Research Journal of Engineering and Technology* 8, 6 (2021), 716–718.

Received 5 January 2024; accepted 29 January 2024; revised 11 March 2024