

AI Benchmark: All About Deep Learning on Smartphones in 2019

Andrey Ignatov

ETH Zurich

andrey@vision.ee.ethz.ch

Radu Timofte

ETH Zurich

timofter@vision.ee.ethz.ch

Andrei Kulik

Google Research

akulik@google.com

Seungsoo Yang

Samsung, Inc.

ssl.yang@samsung.com

Ke Wang

Huawei, Inc.

michael.wangke@huawei.com

Felix Baum

Qualcomm, Inc.

fbaum@qti.qualcomm.com

Max Wu

MediaTek, Inc.

max.wu@mediatek.com

Lirong Xu

Unisoc, Inc.

lirong.Xu@unisoc.com

Luc Van Gool*

ETH Zurich

vangool@vision.ee.ethz.ch

Abstract

The performance of mobile AI accelerators has been evolving rapidly in the past two years, nearly doubling with each new generation of SoCs. The current 4th generation of mobile NPUs is already approaching the results of CUDA-compatible Nvidia graphics cards presented not long ago, which together with the increased capabilities of mobile deep learning frameworks makes it possible to run complex and deep AI models on mobile devices. In this paper, we evaluate the performance and compare the results of all chipsets from Qualcomm, HiSilicon, Samsung, MediaTek and Unisoc that are providing hardware acceleration for AI inference. We also discuss the recent changes in the Android ML pipeline and provide an overview of the deployment of deep learning models on mobile devices. All numerical results provided in this paper can be found and are regularly updated on the official project website¹.

1. Introduction

Over the past years, deep learning and AI became one of the key trends in the mobile industry. This was a natural fit, as from the end of the 90s mobile devices were getting equipped with more and more software for intelligent data processing – face and eyes detection [20], eye tracking [53], voice recognition [51], barcode scanning [84], accelerometer-based gesture recognition [48, 57], predictive text recognition [74], handwritten text recognition [4], OCR [36], etc. At the beginning, all proposed methods were mainly based on manually designed features and very

compact models as they were running at best on devices with a single-core 600 MHz Arm CPU and 8-128 MB of RAM. The situation changed after 2010, when mobile devices started to get multi-core processors, as well as powerful GPUs, DSPs and NPUs, well suitable for machine and deep learning tasks. At the same time, there was a fast development of the deep learning field, with numerous novel approaches and models that were achieving a fundamentally new level of performance for many practical tasks, such as image classification, photo and speech processing, neural language understanding, etc. Since then, the previously used hand-crafted solutions were gradually replaced by considerably more powerful and efficient deep learning techniques, bringing us to the current state of AI applications on smartphones.

Nowadays, various deep learning models can be found in nearly any mobile device. Among the most popular tasks are different computer vision problems like image classification [38, 82, 23], image enhancement [27, 28, 32, 30], image super-resolution [17, 42, 83], bokeh simulation [85], object tracking [87, 25], optical character recognition [56], face detection and recognition [44, 70], augmented reality [3, 16], etc. Another important group of tasks running on mobile devices is related to various NLP (Natural Language Processing) problems, such as natural language translation [80, 7], sentence completion [52, 24], sentence sentiment analysis [77, 72, 33], voice assistants [18] and interactive chatbots [71]. Additionally, many tasks deal with time series processing, e.g., human activity recognition [39, 26], gesture recognition [60], sleep monitoring [69], adaptive power management [50, 47], music tracking [86] and classification [73]. Lots of machine and deep learning algorithms are also integrated directly into smartphones firmware and used as auxiliary methods for estimating various parameters and for intelligent data processing.

*We also thank Oli Gaymond (ogaymond@google.com), Google Inc., for writing and editing section 3.1 of this paper.

¹<http://ai-benchmark.com>

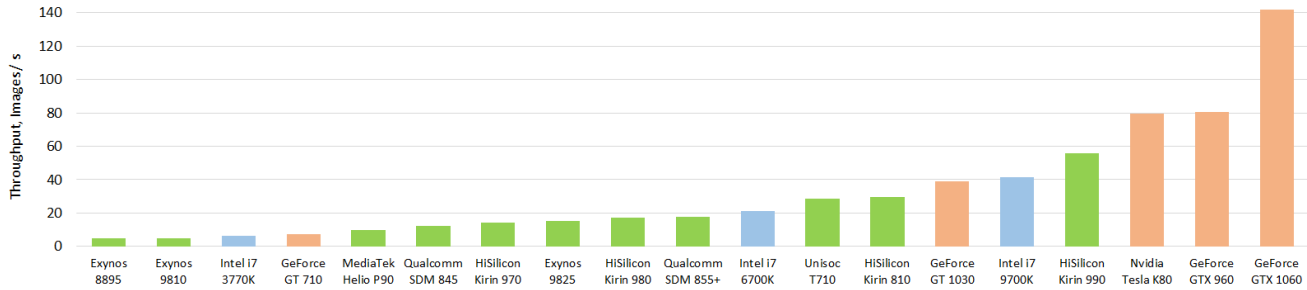


Figure 1: Performance evolution of mobile AI accelerators: image throughput for the float Inception-V3 model. Mobile devices were running the FP16 model using TensorFlow Lite and NNAPI. Acceleration on Intel CPUs was achieved using the Intel MKL-DNN library [45], on Nvidia GPUs – with CUDA [10] and cuDNN [8]. The results on Intel and Nvidia hardware were obtained using the standard TensorFlow library [2] running the FP32 model with a batch size of 20 (the FP16 format is currently not supported by these CPUs / GPUs). Note that the Inception-V3 is a relatively small network, and for bigger models the advantage of Nvidia GPUs over other silicon might be larger.

While running many state-of-the-art deep learning models on smartphones was initially a challenge as they are usually not optimized for mobile inference, the last few years have radically changed this situation. Presented back in 2015, TensorFlow Mobile [79] was the first official library allowing to run standard AI models on mobile devices without any special modification or conversion, though also without any hardware acceleration, i.e. on CPU only. In 2017, the latter limitation was lifted by the TensorFlow Lite (TFLite) [46] framework that dropped support for many vital deep learning operations, but offered a significantly reduced binary size and kernels optimized for on-device inference. This library also got support for the Android Neural Networks API (NNAPI) [5], introduced the same year and allowing for the access to the device’s AI hardware acceleration resources directly through the Android operating system. This was an important milestone as a full-fledged mobile ML pipeline was finally established: training, exporting and running the resulting models on mobile devices became possible within one standard deep learning library, without using specialized vendors tools or SDKs. At first, however, this approach had also numerous flaws related to NNAPI and TensorFlow Lite themselves, thus making it impractical for many use cases. The most notable issues were the lack of valid NNAPI drivers in the majority of Android devices (only 4 commercial models featured them as of September 2018 [19]), and the lack of support for many popular ML models by TFLite. These two issues were largely resolved during the past year. Since the spring of 2019, nearly all new devices with Qualcomm, HiSilicon, Samsung and MediaTek systems on a chip (SoCs) and with dedicated AI hardware are shipped with NNAPI drivers allowing to run ML workloads on embedded AI accelerators. In Android 10, the Neural Networks API was upgraded to version 1.2 that implements 60 new ops [1] and extends the range of supported models. Many of these ops were also added to TensorFlow Lite starting from builds 1.14 and 1.15. Another important change was the in-

roduction of TFLite delegates [12]. These delegates can be written directly by hardware vendors and then used for accelerating AI inference on devices with outdated or absent NNAPI drivers. A universal delegate for accelerating deep learning models on mobile GPUs (based on OpenGL ES, OpenCL or Metal) was already released by Google earlier this year [43]. All these changes build the foundation for a new mobile AI infrastructure tightly connected with the standard machine learning (ML) environment, thus making the deployment of machine learning models on smartphones easy and convenient. The above changes will be described in detail in Section 3.

The latest generation of mid-range and high-end mobile SoCs comes with AI hardware, the performance of which is getting close to the results of desktop CUDA-enabled Nvidia GPUs released in the past years. In this paper, we present and analyze performance results of all generations of mobile AI accelerators from Qualcomm, HiSilicon, Samsung, MediaTek and Unisoc, starting from the first mobile NPUs released back in 2017. We compare against the results obtained with desktop GPUs and CPUs, thus assessing performance of mobile vs. conventional machine learning silicon. To do this, we use a professional AI Benchmark application [31] consisting of 21 deep learning tests and measuring more than 50 different aspects of AI performance, including the speed, accuracy, initialization time, stability, etc. The benchmark was significantly updated since previous year to reflect the latest changes in the ML ecosystem. These updates are described in Section 4. Finally, we provide an overview of the performance, functionality and usage of Android ML inference tools and libraries, and show the performance of more than 200 Android devices and 100 mobile SoCs collected in-the-wild with the AI Benchmark application.

The rest of the paper is arranged as follows. In Section 2 we describe the hardware acceleration resources available on the main chipset platforms and programming interfaces

to access them. Section 3 gives an overview of the latest changes in the mobile machine learning ecosystem. Section 4 provides a detailed description of the recent modifications in our AI Benchmark architecture, its programming implementation and deep learning tests. Section 5 shows the experimental performance results for various mobile devices and chipsets, and compares them to the performance of desktop CPUs and GPUs. Section 6 analyzes the results. Finally, Section 7 concludes the paper.

2. Hardware Acceleration

Though many deep learning algorithms were presented back in the 1990s [40, 41, 22], the lack of appropriate (and affordable) hardware to train such models prevented them from being extensively used by the research community till 2009, when it became possible to effectively accelerate their training with general-purpose consumer GPUs [65]. With the introduction of Max-Pooling CNNs [9, 55] and AlexNet [38] in 2011-2012 and the subsequent success of deep learning in many practical tasks, it was only a matter of time before deep neural networks would be run on mobile devices. Compared to simple statistical methods previously deployed on smartphones, deep learning models required huge computational resources and thus running them on Arm CPUs was nearly infeasible from both the performance and power efficiency perspective. The first attempts to accelerate AI models on mobile GPUs and DSPs were made in 2015 by Qualcomm [89], Arm [58] and other SoC vendors, though at the beginning mainly by adapting deep learning models to the existing hardware. Specialized AI silicon started to appear in mobile SoCs with the release of the Snapdragon 820 / 835 with the Hexagon V6 68x DSP series optimized for AI inference, the Kirin 970 with a dedicated NPU unit designed by Cambricon, the Exynos 8895 with a separate Vision Processing Unit, MediaTek Helio P60 with AI Processing Unit, and the Google Pixel 2 with a standalone Pixel Visual Core. The performance of mobile AI accelerators has been evolving extremely rapidly in the past three years (Fig. 1), coming ever closer to the results of desktop hardware. We can now distinguish four generations of mobile SoCs based on their AI performance, capabilities and release date:

Generation 1: All legacy chipsets that can not provide AI acceleration through the Android operating system, but still can be used to accelerate machine learning inference with special SDKs or GPU-based libraries. All Qualcomm SoCs with Hexagon 682 DSP and below, and the majority of chipsets from HiSilicon, Samsung and MediaTek belong to this category. It is worth mentioning that nearly all computer vision models are largely based on vector and matrix multiplications, and thus can technically run on almost any mobile GPU supporting OpenGL ES or OpenCL. Yet, this approach might actually lead to notable performance degradation on many SoCs with low-end or old-gen GPUs.

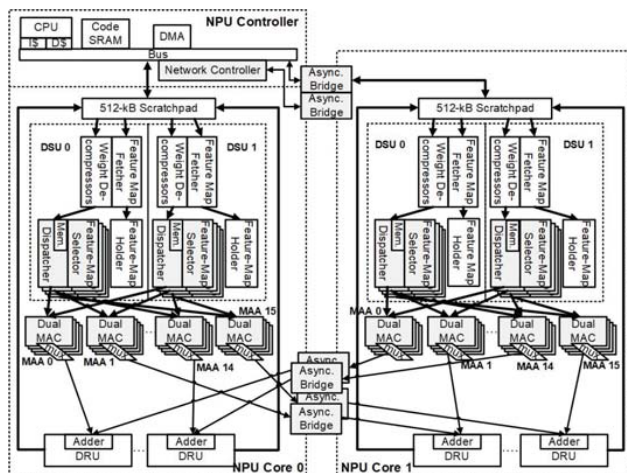


Figure 2: The overall architecture of the Exynos 9820 NPU [78].

Generation 2: Mobile SoCs supporting Android NNAPI and released after 2017. They might provide acceleration for only one type of models (float or quantized) and are typical for the AI performance in 2018.

- Qualcomm: Snapdragon 845 (Hex. 685 + Adreno 630);
Snapdragon 710 (Hexagon 685);
Snapdragon 670 (Hexagon 685);
- HiSilicon: Kirin 970 (NPU, Cambricon);
- Samsung: Exynos 9810 (Mali-G72 MP18);
Exynos 9610 (Mali-G72 MP3);
Exynos 9609 (Mali-G72 MP3);
- MediaTek: Helio P70 (APU 1.0 + Mali-G72 MP3);
Helio P60 (APU 1.0 + Mali-G72 MP3);
Helio P65 (Mali-G52 MP2).

Generation 3. Mobile SoCs supporting Android NNAPI and released after 2018. They provide hardware acceleration for all model types and their AI performance is typical for the corresponding SoC segment in 2019.

- Qualcomm: Snapdragon 855+ (Hex. 690 + Adreno 640);
Snapdragon 855 (Hex. 690 + Adreno 640);
Snapdragon 730 (Hex. 688 + Adreno 618);
Snapdragon 675 (Hex. 685 + Adreno 612);
Snapdragon 665 (Hex. 686 + Adreno 610);
- HiSilicon: Kirin 980 (NPU×2, Cambricon);
- Samsung: Exynos 9825 (NPU + Mali-G76 MP12);
Exynos 9820 (NPU + Mali-G76 MP12);
- MediaTek: Helio P90 (APU 2.0);
Helio G90 (APU 1.0 + Mali-G76 MP4).

Generation 4: Recently presented chipsets with next-generation AI accelerators (Fig. 1). Right now, only the HiSilicon Kirin 990, HiSilicon Kirin 810 and Unisoc Tiger T710 SoCs belong to this category. Many more chipsets from other vendors will come by the end of this year.

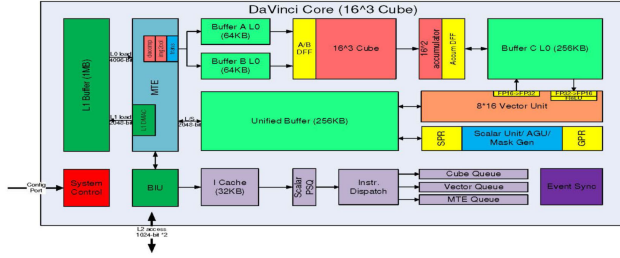


Figure 3: A general architecture of the Huawei's DaVinci Core.

Below, we provide a detailed description of the mobile platforms and related SDKs released in the past year. More information about SoCs with AI acceleration support that were introduced earlier, can be found in our previous paper [31].

2.1. Samsung chipsets / EDEN SDK

The Exynos 9820 was the first Samsung SoC to get an NPU technically compatible with Android NNAPI, its drivers will be released after Android Q upgrade. This chipset contains two custom Mongoose M4 CPU cores, two Cortex-A75, four Cortex-A55 cores and Mali-G76 MP12 graphics. The NPU of the Exynos 9820 supports only quantized inference and consists of the controller and two cores (Fig. 2) having 1024 multiply-accumulate (MAC) units [78]. The NPU controller has a CPU, a direct memory access (DMA) unit, code SRAM and a network controller. The CPU is communicating with the host system of the SoC and defines the network scale for the network controller. The controller automatically configures all modules in the two cores and traverses the network. To use the external memory bandwidth and the scratchpads efficiently, the weights of the network are compressed, and the network compiler additionally partitions the network into sub-networks and performs the traversal over multiple network layers. The DMA unit manages the compressed weights and feature maps in each of the 512KB scratchpads of the cores. When running the computations, the NPU can also skip weights that are zero to improve convolution efficiency. A much more detailed description of the Exynos NPU can be found in [78]. We strongly recommend reading this article for everyone interested in the general functioning of NPUs as it provides an excellent overview on all network / data processing stages and possible bottlenecks.

The Exynos 9820's NPU occupies 5.5mm², is fabricated in 8nm CMOS technology and operates at 67-933 MHz clock frequency. The performance of the NPU heavily depends on the kernel sizes and the fraction of zero weights. For kernels of size 5×5, it achieves the performance of 2.1 TOPS and 6.9 TOPS for 0% and 75% zero-weights, respectively; the energy efficiency in these two cases is 3.6 TOPS/W and 11.5 TOPS/W. For the Inception-V3

	Kirin 990 5G	Kirin 990
Chipset Process	7nm+ EUV	7nm
CPU	2X Cortex-A76 Based @2.86GHz 2X Cortex-A76 Based @2.36GHz 4X Cortex-A55 @1.95GHz	2X Cortex-A76 Based @2.86GHz 2X Cortex-A76 Based @2.09GHz 4X Cortex-A55 @1.86GHz
GPU	16 Core Mali-G76	16 Core Mali-G76
NPU	2 Big Core +1 Micro Core	1 Big Core +1 Micro Core
UFS	UFS 3.0, UFS 2.1	UFS 3.0, UFS 2.1
Modem	2G/3G/4G/5G	2G/3G/4G

Figure 4: SoC components integrated into the Kirin 990 chips.

model, the energy efficiency lies between 2 TOPS/W and 3.4 TOPS/W depending on network sparsity [78].

The other two Samsung SoCs that support Android NNAPI are the Exynos 9609 / 9610, though they are relying on the Mali-G72 MP3 GPU and Arm NN drivers [6] to accelerate AI models. As to the Exynos 9825 presented together with the latest Note10 smartphone series, this is a slightly overclocked version of the Exynos 9820 produced in 7nm technology, with the same NPU design.

This year, Samsung announced the Exynos Deep Neural Network (EDEN) SDK that provides the NPU, GPU and CPU acceleration for deep learning models and exploits the data and model parallelism. It consists of the model conversion tool, the NPU compiler and the customized TFLite generator and is available as a desktop tool plus runtimes for Android and Linux. The EDEN runtime provides APIs for initialization, opening / closing the model and its execution with various configurations. Unfortunately, it is not publicly available yet.

2.2. HiSilicon chipsets / HiAI SDK

While the Kirin 970 / 980 SoCs were using NPUs originally designed by Cambricon, this year Huawei switched to its in-house developed Da Vinci architecture (Fig. 3), powering the Ascend series of AI accelerators and using a 3D Cube computing engine to accelerate matrix computations. The first SoC with Da Vinci NPU was a mid-range Kirin 810 incorporating two Cortex-A76 and six Cortex-A55 CPU cores with Mali-G52 MP6 GPU. A significantly enlarged AI accelerator appeared later in the Kirin 990 5G chip having four Cortex-A76, four Cortex-A55 CPUs and Mali-G76 MP16 graphics. This SoC features a triple-core Da Vinci NPU containing two large (Da Vinci Lite) cores for heavy computing scenarios and one little (Da Vinci Tiny) core for low-power AI computations. According to Huawei, the little core is up to 24 times more power efficient than the large one when running face recognition models. Besides that, a simplified version of the Kirin 990 (without “5G” prefix) with a dual-core NPU (one large + one small core) was also presented and should not be confused with the standard version (Fig. 4).

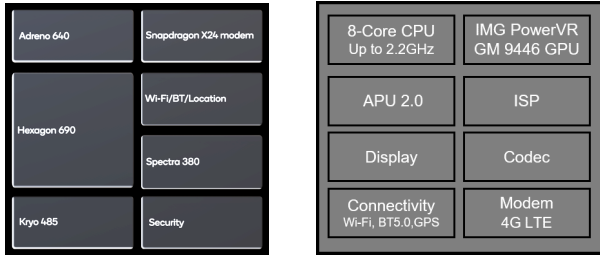


Figure 5: Qualcomm Snapdragon 855 (left) and MediaTek Helio P90 (right) block diagrams.

In the late 2018, Huawei launched the HiAI 2.0 SDK with added support for the Kirin 980 chipset and new deep learning ops. Huawei has also released the IDE tool and Android Studio plug-in, providing development toolsets for running deep learning models with the HiAI Engine. With the recent update of HiAI, it supports more than 300 deep learning ops and the latest Kirin 810 / 990 (5G) SoCs.

2.3. Qualcomm chipsets / SNPE SDK

As before, Qualcomm is relying on its AI Engine (consisting of the Hexagon DSP, Adreno GPU and Kryo CPU cores) for the acceleration of AI inference. In all Qualcomm SoCs supporting Android NNAPI, the Adreno GPU is used for floating-point deep learning models, while the Hexagon DSP is responsible for quantized inference. It should be noted that though the Hexagon 68x/69x chips are still marketed as DSPs, their architecture was optimized for deep learning workloads and they include dedicated AI silicon such as tensor accelerator units, thus not being that different from NPUs and TPUs proposed by other vendors. The only major weakness of the Hexagon DSPs is the lack of support for floating-point models (same as in the Google Pixel TPU, MediaTek APU 1.0 and Exynos NPU), thus the latter are delegated to Adreno GPUs.

At the end of 2018, Qualcomm announced its flagship SoC, the Snapdragon 855, containing eight custom Kryo 485 CPU cores (three clusters functioning at different frequencies, Cortex-A76 derived), an Adreno 640 GPU and Hexagon 690 DSP (Fig. 5). Compared to the Hexagon 685 used in the SDM845, the new DSP got a 1024-bit SIMD with double the number of pipelines and an additional tensor accelerator unit. Its GPU was also upgraded from the previous generation, getting twice more ALUs and an expected performance increase of 20% compared to the Adreno 630. The Snapdragon 855 Plus, released in July 2019, is an over-clocked version of the standard SDM855 SoC, with the same DSP and GPU working at higher frequencies. The other three mid-range SoCs introduced in the past year (Snapdragon 730, 665 and 675) include the Hexagon 688, 686 and 685 DSPs, respectively (the first two are derivatives of the Hexagon 685). All the above mentioned SoCs support

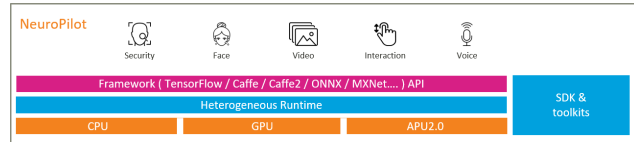


Figure 6: Schematic representation of MediaTek NeuroPilot SDK.

Android NNAPI 1.1 and provide acceleration for both float and quantized models. According to Qualcomm, all NNAPI-compliant chipsets (Snapdragon 855, 845, 730, 710, 675, 670 and 665) will get support for NNAPI 1.2 in Android Q.

Qualcomm’s Neural Processing SDK (SNPE) [76] also went through several updates in the past year. It currently offers Android and Linux runtimes for neural network model execution, APIs for controlling loading / execution / scheduling on the runtimes, desktop tools for model conversion and a performance benchmark for bottleneck identification. It currently supports the Caffe, Caffe2, ONNX and TensorFlow machine learning frameworks.

2.4. MediaTek chipsets / NeuroPilot SDK

One of the key releases from MediaTek in the past year was the Helio P90 with a new AI Processing Unit (APU 2.0) that can generate a computational power of up to 1.1 TMACs / second (4 times higher than the previous Helio P60 / P70 series). The SoC, manufactured with a 12nm process, combines a pair of Arm Cortex-A75 and six Cortex-A55 CPU cores with the IMG PowerVR GM 9446 GPU and dual-Channel LPDDR4x RAM up to 1866MHz. The design of the APU was optimized for operations intensively used in deep neural networks. First of all, its parallel processing engines are capable of accelerating heavy computing operations, such as convolutions, fully connected layers, activation functions, 2D operations (*e.g.*, pooling or bilinear interpolation) and other tensor manipulations. The task control system and data buffer were designed to minimize memory bandwidth usage and to maximize data reuse and the utilization rate of processing engines. Finally, the APU is supporting all popular inference modes, including FP16, INT16 and INT8, allowing to run all common AI models with hardware acceleration. Taking face detection as an example, the APU can run up to 20 times faster and reduce the power consumption by 55 times compared to the Helio’s CPU.

As to other MediaTek chipsets presented this year, the Helio G90 and the Helio P65 are also providing hardware acceleration for float and quantized AI models. The former uses a separate APU (1st gen.) with a similar architecture as the one in the Helio P60 / P70 chipsets [31]. The Helio P65 does not have a dedicated APU module and is running all models on a Mali-G52 MP2 GPU.

Together with the Helio P90, MediaTek has also launched the NeuroPilot v2.0 SDK (Fig. 6). In its second ver-

sion, NeuroPilot supports automatic network quantization and pruning. The SDK’s APU drivers support FP16/INT16/INT8 data types, while CPU and GPU drivers can be used for some custom ops and FP32/FP16 models. The NeuroPilot SDK was designed to take advantage of MediaTek’s heterogeneous hardware, by assigning the workloads to the most suitable processor and concurrently utilizing all available computing resources for the best performance and energy efficiency. The SDK is supporting only MediaTek NeuroPilot-compatible chipsets across products such as smartphones and TVs. At its presentation of the Helio P90, MediaTek demonstrated that NeuroPilot v2.0 allows for the real-time implementation of many AI applications (e.g. multi-person pose tracking, 3D pose tracking, multiple object identification, AR / MR, semantic segmentation, scene identification and image enhancement).

2.5. Unisoc chipsets / UNIAI SDK

Unisoc is a Chinese fabless semiconductor company (formerly known as Spreadtrum) founded in 2001. The company originally produced chips for GSM handsets and was mainly known in China, though starting from 2010-2011 it began to expand its business to the global market. Unisoc’s first smartphone SoCs (SC8805G and SC6810) appeared in entry-level Android devices in 2011 and were featuring an ARM-9 600MHz processor and 2D graphics. With the introduction of the quad-core Cortex-A7 based SC773x, SC883x and SC983x SoC series, Unisoc chipsets became used in many low-end, globally shipped Android devices. The performance of Unisoc’s budget chips was notably improved in the SC9863 SoC and in the Tiger T310 platform released earlier this year. To target the mid-range segment, Unisoc introduced the Tiger T710 SoC platform with four Cortex-A75 + four Cortex-A55 CPU cores and IMG PowerVR GM 9446 graphics. This is the first chipset from Unisoc to feature a dedicated NPU module for the acceleration of AI computations. The NPU of the T710 consists of two different computing accelerator cores: one for integer models supporting the INT4, INT8 and INT16 formats and providing a peak performance of 3.2 TOPS for INT8, and the other for FP16 models with 0.5 TFLOPS performance. The two cores can either accelerate different AI tasks at the same time, or accelerate the task with one of them, while the second core can be completely shut down to reduce the overall power consumption of the SoC. The Tiger T710 supports Android NNAPI and implements Android NN Unosic HIDL services supporting INT8 / FP16 models. The overall energy efficiency of the T710’s NPU is greater than or equal to 2.5 TOPS/W depending on the scenario.

Unisoc has also developed the UNIAI SDK 7 that consists of two parts: the off-line model conversion tool that can compile the trained model into a file that can be executed on NPU; and the off-line model API and runtime used to

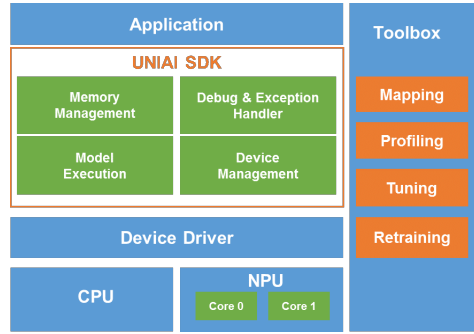


Figure 7: Schematic representation of Unisoc UNIAI SDK.

load and execute the compiled model. The off-line model conversion tool supports several neural network framework formats, including Tensorflow, Tensorflow Lite, Caffe and ONNX. To improve the flexibility, the NPU Core also includes units that can be programmed to support user defined ops, making it possible to run the entire model with such ops on NPU and thus significantly decreasing runtime.

2.6. Google Pixel 3 / Pixel Visual Core

As for the Pixel 2 series, the third generation of Google phones contains a separate tensor processing unit (Pixel Visual Core) capable of accelerating deep learning ops. This TPU did not undergo significant design changes compared to the previous version. Despite Google’s initial statement [66], neither SDK nor NNAPI drivers were or will be released for this TPU series, making it inaccessible to anyone except Google. Therefore, its importance for deep learning developers is limited. In the Pixel phones, it is used for a few tasks related to HDR photography and real-time sensor data processing.

3. Deep Learning on Smartphones

In a preceding paper ([31], Section 3), we described the state of the deep learning mobile ecosystem as of September 2018. The changes in the past year were along the line of expectations. The TensorFlow Mobile [79] framework was completely deprecated by Google in favor of TensorFlow Lite that got a significantly improved CPU backend and support for many new ops. Yet, TFLite is still lacking some vital deep learning operators, especially those used in many NLP models. Therefore, TensorFlow Mobile remains relevant for complex architectures. Another recently added option for unsupported models is to use the TensorFlow Lite plugin containing standard TensorFlow operators [63] that are not yet added to TFLite. That said, the size of this plugin (40MB) is even larger than the size of the TensorFlow Mobile library (20MB). As to the Caffe2 / PyTorch libraries, while some unofficial Android ports appeared in the past 12

months [64, 13], there is still no official support for Android (except for 2 two-year old camera demos [15, 14]), thus making it not that interesting for regular developers.

Though some TensorFlow Lite issues mentioned last year [31] were solved in its current releases, we still recommend using it with great precaution. For instance, in its latest official build (1.14), the interaction with NNAPI was completely broken, leading to enormous losses and random outputs during the first two inferences. This issue can be solved by replacing the `setUseNNAPI` method with a stand-alone NNAPI delegate present in the TFLite-GPU delegate library [11]. Another problem present in the nightly builds is a significantly increased RAM consumption for some models (e.g., SRCNN, Inception-ResNet-V1, VGG-19), making them crashing even on devices with 4GB+ of RAM. While these issues should be solved in the next official TFLite release (1.15), we suggest developers to extensively test their models on all available devices with each change of TFLite build. Another recommended option is to move to custom TensorFlow Lite delegates from SoC vendors that allow to omit such problems and potentially achieve even better results on their hardware.

The other two major changes in the Android deep learning ecosystem were the introduction of TensorFlow Lite delegates and Neural Networks API 1.2. We describe them in detail below.

3.1. Android NNAPI 1.2

The latest version of NN API provides access to 56 new operators, significantly expanding the range of models that can be supported for hardware acceleration. In addition the range of supported data types has increased, bringing support for per-axis quantization for weights and IEEE Float 16. This broader support for data types enables developers and hardware makers to determine the most performant options for their specific model needs.

A significant addition to the API surface is the ability to query the underlying hardware accelerators at runtime and specify explicitly where to run the model. This enables use cases where the developer wants to limit contention between resources, for example an Augmented Reality developer may choose to ensure the GPU is free for visual processing requirements by directing their ML workloads to an alternative accelerator if available.

Neural Networks API 1.2 introduces the concept of burst executions. Burst executions are a sequence of executions of the same prepared model that occur in rapid succession, such as those operating on frames of a camera capture or successive audio samples. A burst object is used to control a set of burst executions, and to preserve resources between executions, enabling executions to have lower overhead. From Android 10, NNAPI provides functions to support caching of compilation artifacts, which reduces the time used for com-

pilation when an application starts. Using this caching functionality, the driver does not need to manage or clean up the cached files. Neural Networks API (NNAPI) vendor extensions, introduced in Android 10, are collections of vendor-defined operations and data types. On devices running NN HAL 1.2 or higher, drivers can provide custom hardware-accelerated operations by supporting corresponding vendor extensions. Vendor extensions do not modify the behavior of existing operations. Vendor extensions provide a more structured alternative to OEM operation and data types, which were deprecated in Android 10.

3.2. TensorFlow Lite Delegates

In the latest releases, TensorFlow Lite provides APIs for delegating the execution of neural network sub-graphs to external libraries (called delegates) [12]. Given a neural network model, TFLite first checks what operators in the model can be executed with the provided delegate. Then TFLite partitions the graph into several sub-graphs, substituting the supported by the delegate sub-graphs with virtual “delegate nodes” [43]. From that point, the delegate is responsible for executing all sub-graphs in the corresponding nodes. Unsupported operators are by default computed by the CPU, though this might significantly increase the inference time as there is an overhead for passing the results from the sub-graph to the main graph. The above logic is already used by the TensorFlow Lite GPU backend described in the next section.

3.3. TensorFlow Lite GPU Delegate

While many different NPUs were already released by all major players, they are still very fragmented due to a missing common interface or API. While NNAPI was designed to tackle this problem, it suffers from its own design flaws that slow down NNAPI adoption and usage growth:

- **Long update cycle:** NNAPI update is still bundled with the OS update. Thus, it may take up to a year to get new drivers.
- **Custom operations support:** When a model has an op that is not yet supported by NNAPI, it is nearly impossible to run it with NNAPI. In the worst case, two parts of a graph are accelerated through NNAPI, while a single op implemented out of the context is computed by the CPU, which ruins the performance.

There is another attempt by the Vulkan ML group to introduce common programming language to be implemented by vendors. The language resembles a model graph representation similar to one found in the TensorFlow or ONNX libraries. The proposal is still in its early stage and, if accepted, will take a few years to reach consumer devices.

Besides the above issues, there also exists a huge fragmentation of mobile hardware platforms. For instance, the most popular 30 SoC designs are now representing only 51% of the market share, while 225 SoCs are still covering just 95% of the market with a long tail of a few thousand designs. The majority of these SoCs will never get NNAPI drivers, though one should mention that around 23% of them have GPUs at least 2 times more performant than the corresponding CPUs, and thus they can be used for accelerating ML inference. This number is significantly bigger than the current market share of chipsets with NPUs or valid NNAPI drivers. To use the GPU acceleration on such platforms, TensorFlow GPU delegate was introduced.

The inference phase of the GPU delegate consists of the following steps. The input tensors are first reshaped to the PHWC4 format if their tensor shape has channel size not equal to 4. For each operator, shader programs are linked by binding resources such the operators input / output tensors, weights, etc. and dispatched, i.e. inserted into the command queue. The GPU driver then takes care of scheduling and executing all shader programs in the queue, and makes the result available to the CPU by the CPU / GPU synchronization. In the GPU inference engine, operators exist in the form of shader programs. The shader programs eventually get compiled and inserted into the command queue and the GPU executes programs from this queue without synchronization with the CPU. After the source code for each program is generated, each shader gets compiled. This compilation step can take awhile, from several milliseconds to seconds. Typically, app developers can hide this latency while loading the model or starting the app for the first time. Once all shader programs are compiled, the GPU backend is ready for inference. A much more detailed description of the TFLite GPU delegate can be found in [43].

3.4. Floating-point vs. Quantized Inference

One of the most controversial topics related to the deployment of deep learning models on smartphones is the suitability of floating-point and quantized models for mobile devices. There has been a lot of confusion with these two types in the mobile industry, including a number of incorrect statements and invalid comparisons. We therefore decided to devote a separate section to them and describe and compare their benefits and disadvantages. We divided the discussion into three sections: the first two are describing each inference type separately, while the last one compares them directly and makes suggestions regarding their application.

3.4.1. Floating-point Inference

Advantages: The model is running on mobile devices in the same format as it was originally trained on the server or desktop with standard machine learning libraries. No special conversion, changes or re-training is needed; thus one

gets the same accuracy and performance as on the desktop or server environment.

Disadvantages: Many recent state-of-the-art deep learning models, especially those that are working with high-resolution image transformations, require more than 6-8 gigabytes of RAM and enormous computational resources for data processing that are not available even in the latest high-end smartphones. Thus, running such models in their original format is infeasible, and they should be first modified to meet the hardware resources available on mobile devices.

3.4.2. Quantized Inference

Advantages: The model is first converted from a 16-bit floating point type to int-8 format. This reduces its size and RAM consumption by a factor of 4 and potentially speeds up its execution by 2-3 times. Since integer computations consume less energy on many platforms, this also makes the inference more power efficient, which is critical in the case of smartphones and other portable electronics.

Disadvantages: Reducing the bit-width of the network weights (from 16 to 8 bits) leads to accuracy loss: in some cases, the converted model might show only a small performance degradation, while for some other tasks the resulting accuracy will be close to zero. Although a number of research papers dealing with network quantization were presented by Qualcomm [49, 54] and Google [34, 37], all showing decent accuracy results for many image classification models, there is no general recipe for quantizing arbitrary deep learning architectures. Thus, quantization is still more of a research topic, without working solutions for many AI-related tasks (e.g., image-to-image mapping or various NLP problems). Besides that, many quantization approaches require the model to be retrained from scratch, preventing the developers from using available pre-trained models provided together with all major research papers.

3.4.3. Comparison

As one can see, there is always a trade-off between using one model type or another: floating-point models will always show better accuracy (since they can be simply initialized with the weights of the quantized model and further trained for higher accuracy), while integer models yield faster inference. The progress here comes from both sides: AI accelerators for floating-point models are becoming faster and are reducing the difference between the speed of INT-8 and FP16 inference, while the accuracy of various network quantization approaches is also rising rapidly. Thus, the applicability of each approach will depend on the particular task and the corresponding hardware / energy consumption limitations: for complex models and high-performance devices float models are preferable (due to the convenience of deployment and better accuracy), while quantized inference is

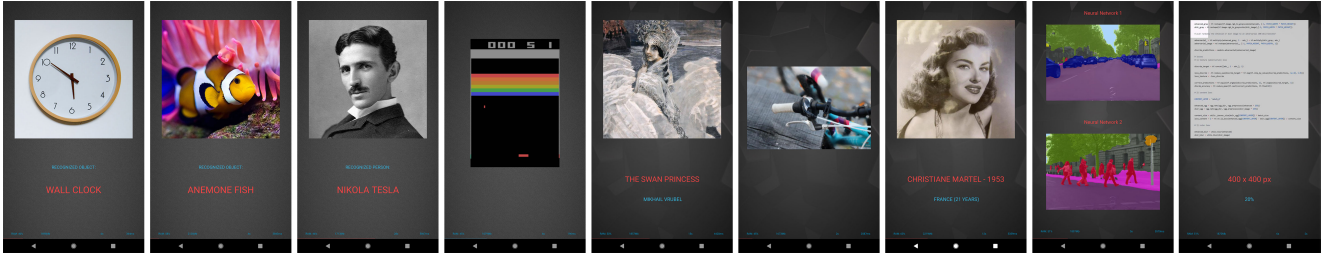


Figure 8: Sample result visualizations displayed to the user in deep learning tests.

clearly beneficial in the case of low-power and low-RAM devices and quantization-friendly models that can be converted from the original float format to INT-8 with a minimal performance degradation.

When comparing float and quantized inference, one good analogy would be the use of FullHD vs. 4K videos on mobile devices. All other things being equal, the latter always have better quality due to their higher resolution, but also demand considerably more disc space or internet bandwidth and hardware resources for decoding them. Besides that, on some screens the difference between 1080P and 4K might not be visible. But this does not mean that one of the two resolutions should be discarded altogether. Rather, the most suitable solution should be used in each case.

Last but not least, one should definitely avoid comparing the performance of two different devices by running floating-point models on one and quantized models on the other. As they have different properties and show different accuracy results, the obtained numbers will make no sense (same as measuring the FPS in a video game running on two devices with different resolutions). This, however, does not refer to the situation when this is done to demonstrate the comparative performance of two inference types, if accompanied by the corresponding accuracy results.

4. AI Benchmark 3.0

The AI Benchmark application was first released in May 2018, with the goal of measuring the AI performance of various mobile devices. The first version (1.0.0) included a number of typical AI tasks and deep learning architectures, and was measuring the execution time and memory consumption of the corresponding AI models. In total, 12 public versions of the AI Benchmark application were released since the beginning of the project. The second generation (2.0.0) was described in detail in the preceding paper [31]. Below we briefly summarize the key changes introduced in the subsequent benchmark releases:

- **2.1.0** (release date: 13.10.2018) — this version brought a number of major changes to AI Benchmark. The total number of tests was increased from 9 to 11. In test 1, MobileNet-V1 was changed to MobileNet-V2 running in three sub-

tests with different inference types: float model on CPU, float model with NNAPI and quantized model with NNAPI. Inception-ResNet-V1 and VGG-19 models from tests 3 and 5, respectively, were quantized and executed with NNAPI. In test 7, ICNet model was running in parallel in two separate threads on CPU. A more stable and reliable category-based scoring system was introduced. Required Android 4.1 and above.

- **2.1.1** (release date: 15.11.2018) — normalization coefficients used in the scoring system were updated to be based on the best results from the actual SoCs generation (Snapdragon 845, Kirin 970, Helio P60 and Exynos 9810). This version also introduced several bug fixes and an updated ranking table. Required Android 4.1 and above.

- **2.1.2** (release date: 08.01.2019) — contained a bug fix for the last memory test (on some devices, it was terminated before the actual RAM exhaustion).

- **3.0.0** (release date: 27.03.2019) — the third version of AI Benchmark with a new modular-based architecture and a number of major updates. The number of tests was increased from 11 to 21. Introduced accuracy checks, new tasks and networks, PRO mode and updated scoring system that are described further in this section.

- **3.0.1** (release date: 21.05.2019) and **3.0.2** (release date: 13.06.2019) — fixed several bugs and introduced new features in the PRO mode.

Since a detailed technical description of AI Benchmark 2.0 was provided in [31], we here mainly focus on the updates and changes introduced by the latest release.

4.1. Deep Learning Tests

The actual benchmark version (3.0.2) consists of 11 test sections and 21 tests. The networks running in these tests represent the most popular and commonly used deep learning architectures that can be currently deployed on smartphones. The description of test configs is provided below.

Test Section 1: Image Classification

Model: MobileNet-V2 [68],

Inference modes: CPU (FP16/32) and NNAPI (INT8 + FP16)

Image resolution: 224×224 px, Test time limit: 20 seconds

Test	1	2	3	4	5	6	7	8	9	10
Task	Classification	Classification	Face Recognition	Playing Atari	Deblurring	Super-Resolution	Super-Resolution	Bokeh Simulation	Segmentation	Enhancement
Architecture	MobileNet-V2	Inception-V3	Inc-ResNet-V1	LSTM RNN	SRCNN	VGG-19	SRGAN (ResNet-16)	U-Net	ICNet	DPED (ResNet-4)
Resolution, px	224×224	346×346	512×512	84×84	384×384	256×256	512×512	128×128	768×1152	128×192
Parameters	3.5M	27.1M	22.8M	3.4M	69K	665K	1.5M	6.6M	6.7M	400K
Size (float), MB	14	95	91	14	0.3	2.7	6.1	27	27	1.6
NNAPI support	yes	yes	yes	yes (1.2+)	yes	yes	yes (1.2+)	yes (1.2+)	yes	yes
CPU-Float	yes	yes	no	yes	no	no	yes	yes	no	no
CPU-Quant	no	no	yes	no	no	no	yes	no	no	no
NNAPI-Float	yes	yes	yes	no	yes	yes	no	no	yes	yes
NNAPI-Quant	yes	yes	yes	no	yes	yes	no	no	no	no

Table 1: Summary of deep learning models used in the AI Benchmark.

Test Section 2: Image Classification

Model: Inception-V3 [82]

Inference modes: CPU (FP16/32) and NNAPI (INT8 + FP16)

Image resolution: 346×346 px, Test time limit: 30 seconds

Test Section 3: Face Recognition

Model: Inception-ResNet-V1 [81]

Inference modes: CPU (INT8) and NNAPI (INT8 + FP16)

Image resolution: 512×512 px, Test time limit: 30 seconds

Test Section 4: Playing Atari

Model: LSTM [22]

Inference modes: CPU (FP16/32)

Image resolution: 84×84 px, Test time limit: 20 seconds

Test Section 5: Image Deblurring

Model: SRCNN 9-5-5 [17]

Inference modes: NNAPI (INT8 + FP16)

Image resolution: 384×384 px, Test time limit: 30 seconds

Test Section 6: Image Super-Resolution

Model: VGG-19 (VDSR) [35]

Inference modes: NNAPI (INT8 + FP16)

Image resolution: 256×256 px, Test time limit: 30 seconds

Test Section 7: Image Super-Resolution

Model: SRGAN [42]

Inference modes: CPU (INT8 + FP16/32)

Image resolution: 512×512 px, Test time limit: 40 seconds

Test Section 8: Bokeh Simulation

Model: U-Net [67]

Inference modes: CPU (FP16/32)

Image resolution: 128×128 px, Test time limit: 20 seconds

Test Section 9: Image Segmentation

Model: ICNet [90]

Inference modes: NNAPI (2 × FP32 models in parallel)

Image resolution: 768×1152 px, Test time limit: 20 seconds

Test Section 10: Image Enhancement

Model: DPED-ResNet [27, 29]

Inference modes: NNAPI (FP16 + FP32)

Image resolution: 128×192 px, Test time limit: 20 seconds

Test Section 11: Memory Test

Model: SRCNN 9-5-5 [17]

Inference modes: NNAPI (FP16)

Image resolution: from 200×200 px to 2000×2000 px

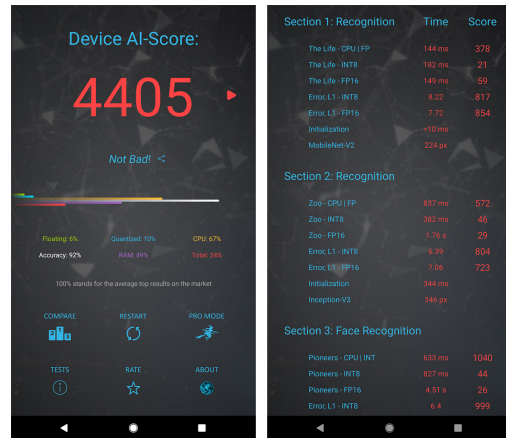


Figure 9: Benchmark results displayed after the end of the tests.

Table 1 summarizes the details of all the deep learning architectures included in the benchmark. When more than one inference mode is used, each image is processed sequentially with all the corresponding modes. In the last memory test, images are processed until the Out-Of-Memory-Error is thrown or all resolutions are processed successfully. In the image segmentation test (Section 9), two TFLite ICNet models are initialized in two separate threads and process images in parallel (asynchronously) in these two threads. The running time for each test is computed as an average over the set of images processed within the specified time. When more than two images are processed, the first two results are discarded to avoid taking into account initialization time (estimated separately), and the average over the rest results is calculated. If less than three images are processed (which happens only on low-end devices), the last inference time is used. The benchmark’s visualization of network outputs is shown in Fig. 8.

Starting from version 3.0.0, AI Benchmark is checking the accuracy of the outputs for float and quantized models running with acceleration (NNAPI) in Test Sections 1, 2, 3, 5 and 6. For each corresponding test, the L_1 loss is computed between the target and actual outputs produced by the deep learning models. The accuracy is estimated separately for both float and quantized models.

4.2. Scoring System

AI Benchmark is measuring the performance of several test categories, including int-8, float-16, float-32, parallel, CPU (int-8 and float-16/32), memory tests, and tests measuring model initialization time. The scoring system used in versions 3.0.0 – 3.0.2 is identical. The contribution of the test categories is as follows:

- 48% - float-16 tests;
- 24% - int-8 tests;
- 12% - CPU, float-16/32 tests;
- 6% - CPU, int-8 tests;
- 4% - float-32 tests;
- 3% - parallel execution of the models;
- 2% - initialization time, float models;
- 1% - initialization time, quantized models;

The scores of each category are computed as a geometric mean of the test results belonging to this category. The computed L1 error is used to penalize the runtime of the corresponding networks running with NNAPI (an exponential penalty with exponent 1.5 is applied). The result of the memory test introduces a multiplicative contribution to the final score, displayed at the end of the tests (Fig. 9). The normalization coefficients for each test are computed based on the best results of the current SoC generation (Snapdragon 855, Kirin 980, Exynos 9820 and Helio P90).

4.3. PRO Mode

The PRO Mode (Fig. 10) was introduced in AI Benchmark 3.0.0 to provide developers and experienced users with the ability to get more detailed and accurate results for tests running with acceleration, and to compare the results of CPU- and NNAPI-based execution for all inference types. It is available only for tasks where both the float and quantized models are compatible with NNAPI (Test Sections 1, 2, 3, 5, 6). In this mode, one can run each of the five inference types (CPU-float, CPU-quantized, float-16-NNAPI, float-32-NNAPI and int-8-NNAPI) to get the following results:

- Average inference time for a single-image inference;
- Average inference time for a throughput inference;
- Standard deviation of the results;
- The accuracy of the produced outputs (L_1 error);
- Model’s initialization time.

Some additional options were added to the PRO Mode in version 3.0.1 that are available under the “Settings” tab:

1. All PRO Mode tests can be run in automatic mode;

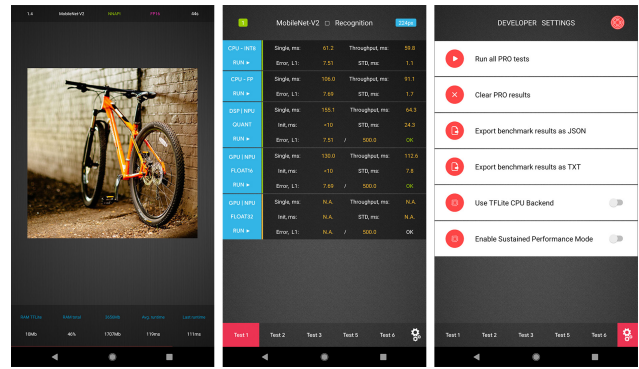


Figure 10: Tests, results and options displayed in the PRO Mode.

2. Benchmark results can be exported to a JSON / TXT file stored in the device’s internal memory;
3. TensorFlow Lite CPU backend can be enabled in all tests for debugging purposes;
4. Sustained performance mode can be used in all tests.

4.4. AI Benchmark for Desktops

Besides the Android version, a separate open source AI Benchmark build for desktops² was released in June 2019. It is targeted at evaluating AI performance of the common hardware platforms, including CPUs, GPUs and TPUs, and measures the inference and training speed for several key deep learning models. The benchmark is relying on the TensorFlow [2] machine learning library and is distributed as a Python pip package³ that can be installed on any system running Windows, Linux or macOS. The current release 0.1.1 consists of 42 tests and 19 sections provided below:

1. MobileNet-V2 [68] [classification]
2. Inception-V3 [82] [classification]
3. Inception-V4 [81] [classification]
4. Inception-ResNet-V2 [21] [classification]
5. ResNet-V2-50 [21] [classification]
6. ResNet-V2-152 [21] [classification]
7. VGG-16 [75] [classification]
8. SRCNN 9-5-5 [17] [image-to-image mapping]
9. VGG-19 [35] [image-to-image mapping]
10. ResNet-SRGAN [42] [image-to-image mapping]
11. ResNet-DPED [27, 29] [image-to-image mapping]
12. U-Net [67] [image-to-image mapping]
13. Nvidia-SPADE [62] [image-to-image mapping]
14. ICNet [90] [image segmentation]

²<http://ai-benchmark.com/alpha>

³<https://pypi.org/project/ai-benchmark>

SoC	AI Accelerator	MobileNet v2, ms	Inception v3, ms	Inc-ResNet v1, ms	SRCNN, ms	VGG-19, ms	DPED, ms	Relative Perf.
HiSilicon Kirin 990	NPU (Da Vinci family)	6	18	37	36	42	19	100%
HiSilicon Kirin 810	NPU (Da Vinci family)	10	34	82	72	122	42	47%
Unisoc Tiger T710	NPU	13	35	80	76	135	43	43%
Snapdragon 855 Plus	GPU (Adreno 640)	15	56	142	66	182	67	32%
HiSilicon Kirin 980	NPU (Cambricon family)	25	58	117	100	163	71	28%
Exynos 9825 Octa	GPU (Mali-G76 MP12, S.LSI OpenCL)	17	65	151	124	158	67	28%
MediaTek Helio P90	APU 2.0	8.3	101	263	75	309	66	26%
Exynos 9820 Octa	GPU (Mali-G76 MP12, S.LSI OpenCL)	19	69	162	137	170	74	26%
Snapdragon 855	GPU (Adreno 640)	24	70	170	87	211	82	25%
Snapdragon 845	GPU (Adreno 630)	27	80	205	98	263	94	21%
HiSilicon Kirin 970	NPU (Cambricon family)	43	69	1514	141	235	83	14%
Snapdragon 730	GPU (Adreno 618)	31	150	391	185	553	175	12%
MediaTek Helio G90T	GPU (Mali-G76 MP4)	37	223	584	459	977	665	6%
Exynos 9820 Octa	GPU (Mali-G76 MP12, Arm NN OpenCL)	40	186	442	889	837	836	6%
Snapdragon 675	GPU (Adreno 612)	39	312	887	523	1238	347	6%
Exynos 9810 Octa	GPU (Mali-G72 MP18)	72	209	488	1574	843	787	4%
Exynos 8895 Octa	GPU (Mali-G71 MP20)	63	216	497	1785	969	909	4%
MediaTek Helio P70	GPU (Mali-G72 MP3)	66	374	932	1096	865	764	4%
MediaTek Helio P65	GPU (Mali-G52 MP2)	51	340	930	752	1675	926	4%
Snapdragon 665	GPU (Adreno 610)	50	483	1292	678	2174	553	4%
MediaTek Helio P60	GPU (Mali-G72 MP3)	68	353	948	1896	889	1439	3%
Exynos 9609	GPU (Mali-G72 MP3)	61	444	1230	1661	1448	731	3%
Exynos 9610	GPU (Mali-G72 MP3)	77	459	1244	1651	1461	773	3%
Exynos 8890 Octa	GPU (Mali-T880 MP12)	98	447	1012	2592	1062	855	3%
Snapdragon 835	None	181	786	1515	1722	3754	1317	1%
GeForce GTX 1080 Ti	CUDA (3584 cores, 1.58 - 1.60 GHz)	1.5	4.5	9.5	4.7	10	4.6	449%
GeForce GTX 950	CUDA (768 cores, 1.02 - 1.19 GHz)	3.9	15	38	23	47	20	115%
Nvidia Tesla K40c	CUDA (2880 cores, 0.75 - 0.88 GHz)	3.7	16	38	22	60	20	111%
Quadro M2000M	CUDA (640 cores, 1.04 - 1.20 GHz)	5	22	54	33	84	30	78%
GeForce GT 1030	CUDA (384 cores, 1.23 - 1.47 GHz)	9.3	31	81	44	97	47	53%
GeForce GT 740	CUDA (384 cores, 0.993 GHz)	12	89	254	238	673	269	14%
GeForce GT 710	CUDA (192 cores, 0.954 GHz)	33	159	395	240	779	249	10%
Intel Core i7-9700K	8/8 @ 3.6 - 4.9 GHz, Intel MKL	4.8	23	72	49	133	72	55%
Intel Core i7-7700K	4/8 @ 4.2 - 4.5 GHz, Intel MKL	7.4	42	121	75	229	100	34%
Intel Core i7-4790K	4/8 @ 4.0 - 4.4 GHz, Intel MKL	8.3	45	133	91	267	124	30%
Intel Core i7-3770K	4/8 @ 3.5 - 3.9 GHz, Intel MKL	12	125	345	209	729	242	13%
Intel Core i7-2600K	4/8 @ 3.4 - 3.8 GHz, Intel MKL	14	143	391	234	816	290	11%
Intel Core i7-950	4/8 @ 3.1 - 3.3 GHz, Intel MKL	36	287	728	448	1219	515	6%

Table 2: Inference time (per one image) for **floating-point networks** obtained on mobile SoCs providing hardware acceleration for fp-16 models. The results of the Snapdragon 835, Intel CPUs and Nvidia GPUs are provided for reference. Acceleration on Intel CPUs was achieved using the Intel MKL-DNN library [45], on Nvidia GPUs – with CUDA [10] and cuDNN [8]. The results on Intel and Nvidia hardware were obtained using the standard TensorFlow library [2] running floating-point models with a batch size of 10. A full list is available at: http://ai-benchmark.com/ranking_processors

15. PSPNet [91] [image segmentation]
16. DeepLab [61] [image segmentation]
17. Pixel-RNN [59] [inpainting]
18. LSTM [22] [sentence sentiment analysis]
19. GNMT [88] [text translation]

The results obtained with this benchmark version are available on the project webpage⁴. Upcoming releases will provide a unified ranking system that allows for a direct comparison of results on mobile devices (obtained with Android AI Benchmark) with those on desktops. The current constraints and particularities of mobile inference do not allow us to merge these two AI Benchmark versions right now, however, they will be gradually consolidated into a single AI Benchmark Suite with a global ranking table. The numbers for desktop GPUs and CPUs shown in the next section were obtained with a modified version of the desktop AI Benchmark build.

⁴http://ai-benchmark.com/ranking_deeplearning

5. Benchmark Results

As the performance of mobile AI accelerators has grown significantly in the past year, we decided to add desktop CPUs and GPUs used for training / running deep learning models to the comparison as well. This will help us to understand how far mobile AI silicon has progressed thus far. It also will help developers to estimate the relation between the runtime of their models on smartphones and desktops. In this section, we present quantitative benchmark results obtained from over 20,000 mobile devices tested in the wild (including a number of prototypes) and discuss in detail the performance of all available mobile chipsets providing hardware acceleration for floating-point or quantized models. The results for floating-point and quantized inference obtained on mobile SoCs are presented in tables 2 and 3, respectively. The detailed performance results for smartphones are shown in table 4.

SoC	AI Accelerator	MobileNet v2, ms	Inception v3, ms	Inc-ResNet v1, ms	SRCNN, ms	VGG-19, ms	Relative Perf.
Snapdragon 855 Plus	DSP (Hexagon 690)	4.9	16	40	24	45	100%
Unisoc Tiger T710	NPU	5	17	38	20	53	99%
HiSilicon Kirin 990	NPU (Da Vinci family)	6.5	20	37	38	39	86%
Snapdragon 855	DSP (Hexagon 690)	8.2	18	46	30	48	80%
MediaTek Helio P90	APU 2.0	4	23	38	22	147	78%
Snapdragon 675	DSP (Hexagon 685)	10	34	73	53	103	47%
Snapdragon 730	DSP (Hexagon 688)	13	47	90	69	111	38%
Snapdragon 670	DSP (Hexagon 685)	12	48	97	153	116	32%
Snapdragon 665	DSP (Hexagon 686)	13	52	118	94	192	29%
Snapdragon 845	DSP (Hexagon 685)	11	45	91	71	608	28%
Exynos 9825 Octa	GPU (Mali-G76 MP12, S.LSI OpenCL)	19	63	128	75	199	27%
Snapdragon 710	DSP (Hexagon 685)	12	48	95	70	607	27%
MediaTek Helio G90T	APU 1.0	15	64	139	107	308	23%
Exynos 9820 Octa	GPU (Mali-G76 MP12, S.LSI OpenCL)	21	73	199	87	262	21%
HiSilicon Kirin 810	NPU (Da Vinci family)	25	98	160	116	172	21%
MediaTek Helio P70	APU 1.0	26	89	181	163	474	15%
MediaTek Helio P60	APU 1.0	27	89	181	164	475	15%
Exynos 9820 Octa	GPU (Mali-G76 MP12, Arm NN OpenCL)	27	96	201	407	446	12%
Exynos 8895 Octa	GPU (Mali-G71 MP20)	44	118	228	416	596	10%
Exynos 9810 Octa	GPU (Mali-G72 MP18)	45	166	360	539	852	7%
MediaTek Helio P65	GPU (Mali-G52 MP2)	43	228	492	591	1167	6%
Snapdragon 835	None	136	384	801	563	1525	3%
Exynos 9609	GPU (Mali-G72 MP3)	50	383	937	1027	2325	3%
Exynos 9610	GPU (Mali-G72 MP3)	52	380	927	1024	2322	3%
Exynos 8890 Octa	GPU (Mali-T880 MP12)	70	378	866	1200	2016	3%

Table 3: Inference time for **quantized networks** obtained on mobile SoCs providing hardware acceleration for int-8 models. The results of the Snapdragon 835 are provided for reference. A full list is available at: http://ai-benchmark.com/ranking_processors

5.1. Floating-point performance

At the end of September 2018, the best publicly available results for floating-point inference were exhibited by the Kirin 970 [31]. The increase in the performance of mobile chips that happened here since that time is dramatic: even without taking into account various software optimizations, the speed of the floating-point execution has increased by more than 7.5 times (from 14% to 100%, table 2). The Snapdragon 855, HiSilicon Kirin 980, MediaTek Helio P90 and Exynos 9820 launched last autumn have significantly improved the inference runtime for float models and already approached the results of several octa-core Intel CPUs (e.g. Intel Core i7-7700K / i7-4790K) and entry-level Nvidia GPUs, while an even higher performance increase was introduced by the 4th generation of AI accelerators released this summer (present in the Unisoc Tiger T710, HiSilicon Kirin 810 and 990). With such hardware, the Kirin 990 managed to get close to the performance of the GeForce GTX 950 – a mid-range desktop graphics card from Nvidia launched in 2015, and significantly outperformed one of the current Intel flagships – an octa-core Intel Core i7-9700K CPU (Coffee Lake family, working frequencies from 3.60 GHz to 4.90 GHz). This is an important milestone as mobile devices are beginning to offer the performance that is sufficient for running many standard deep learning models, even without any special adaptations or modifications. And while this might not be that noticeable in the case of simple image classification networks (MobileNet-V2 can demonstrate 10+ FPS even on

Exynos 8890), it is especially important for various image and video processing models that are usually consuming excessive computational resources.

An interesting topic is to compare the results of GPU- and NPU-based approaches. As one can see, in the third generation of deep learning accelerators (present in the Snapdragon 855, HiSilicon Kirin 980, MediaTek Helio P90 and Exynos 9820 SoCs), they are showing roughly the same performance, while the Snapdragon 855 Plus with an over-clocked Adreno 640 GPU is able to outperform the rest of the chipsets by around 10-15%. However, it is unclear if the same situation will persist in the future: to reach the performance level of the 4th generation NPUs, the speed of AI inference on GPUs should be increased by 2-3 times. This cannot be easily done without introducing some major changes to their micro-architecture, which will also affect the entire graphics pipeline. It therefore is likely that all major chip vendors will switch to dedicated neural processing units in the next SoC generations.

Accelerating deep learning inference with the mid-range (e.g., Mali-G72 / G52, Adreno 610 / 612) or old-generation (e.g., Mali-T880) GPUs is not very efficient in terms of the resulting speed. Even worse results will be obtained on the entry-level GPUs since they come with additional computational constraints. One should, however, note that the power consumption of GPU inference is usually 2 to 4 times lower than the same on the CPU. Hence this approach might still be advantageous in terms of overall energy efficiency.

One last thing that should be mentioned here is the performance of the default Arm NN OpenCL drivers. Unfortunately, they cannot unleash the full potential of Mali GPUs, which results in atypically high inference times compared to GPUs with a similar GFLOPS performance (*e.g.*, the Exynos 9820, 9810 or 8895 with Arm NN OpenCL). By switching to their custom vendor implementation, one can achieve up to 10 times speed-up for many deep learning architectures: *e.g.*, the overall performance of the Exynos 9820 with Mali-G76 MP12 rose from 6% to 26% when using Samsung’s own OpenCL drivers. The same also applies to Snapdragon SoCs which NNAPI drivers are based on Qualcomm’s modified OpenCL implementation.

5.2. Quantized performance

This year, the performance ranking for quantized inference (table 3) is led by the Hexagon-powered Qualcomm Snapdragon 855 Plus chipset accompanied by the Unisoc Tiger T710 with a stand-alone NPU. These two SoCs are showing nearly identical results in all int-8 tests, and are slightly (15-20%) faster than the Kirin 990, Helio P90 and the standard Snapdragon 855. As claimed by Qualcomm, the performance of the Hexagon 690 DSP has approximately doubled over the previous-generation Hexagon 685. The latter, together with its derivatives (Hexagon 686 and 688), is currently present in Qualcomm’s mid-range chipsets. One should note that there exist multiple revisions of the Hexagon 685, as well as several versions of its drivers. Hence, the performance of the end devices and SoCs with this DSP might vary quite significantly (*e.g.*, , Snapdragon 675 vs. Snapdragon 845).

As mobile GPUs are primarily designed for floating-point computations, accelerating quantized AI models with them is not very efficient in many cases. The best results were achieved by the Exynos 9825 with Mali-G76 MP12 graphics and custom Samsung OpenCL drivers. It showed an overall performance similar to that of the Hexagon 685 DSP (in the Snapdragon 710), though the inference results of both chips are heavily dependent on the running model. Exynos mid-range SoCs with Mali-G72 MP3 GPU were not able to outperform the CPU of the Snapdragon 835 chipset, similar to the Exynos 8890 with Mali-T880 MP12 graphics. An even bigger difference will be observed for the CPUs from the more recent mobile SoCs. As a result, using GPUs for quantized inference on the mid-range and low-end devices might be reasonable only to achieve a higher power efficiency.

6. Discussion

The tremendous progress in mobile AI hardware since last year [31] is undeniable. When compared to the second generation of NPUs (*e.g.*, the ones in the Snapdragon 845 and Kirin 970 SoCs), the speed of floating-point and quantized

inference has increased by more than 7.5 and 3.5 times, respectively, bringing the AI capabilities of smartphones to a substantially higher level. All flagship SoCs presented during the past 12 months show a performance equivalent to or higher than that of entry-level CUDA-enabled desktop GPUs and high-end CPUs. The 4th generation of mobile AI silicon yields even better results. This means that in the next two-three years all mid-range and high-end chipsets will get enough power to run the vast majority of standard deep learning models developed by the research community and industry. This, in turn, will result in even more AI projects targeting mobile devices as the main platform for machine learning model deployment.

When it comes to the software stack required for running AI algorithms on smartphones, progress here is evolutionary rather than revolutionary. There is still only one major mobile deep learning library, TensorFlow Lite, providing a reasonably high functionality and ease of deployment of deep learning models on smartphones, while also having a large community of developers. This said, the number of critical bugs and issues introduced in its new versions prevents us from recommending it for any commercial projects or projects dealing with non-standard AI models. The recently presented TensorFlow Lite delegates can be potentially used to overcome the existing issues, and besides that allow the SoC vendors to bring AI acceleration support to devices with outdated or absent NNAPI drivers. We also strongly recommend researchers working on their own AI engines to design them as TFLite delegates, as this is the easiest way to make them available for all TensorFlow developers, as well as to make a direct comparison against the current TFLite’s CPU and GPU backends. We hope that more working solutions and mobile libraries will be released in the next year, making the deployment of deep learning models on smartphones a trivial routine.

As before, we plan to publish regular benchmark reports describing the actual state of AI acceleration on mobile devices, as well as changes in the machine learning field and the corresponding adjustments made in the benchmark to reflect them. The latest results obtained with the AI Benchmark and the description of the actual tests is updated monthly on the project website: <http://ai-benchmark.com>. Additionally, in case of any technical problems or some additional questions you can always contact the first two authors of this paper.

7. Conclusions

In this paper, we discussed the latest advances in the area of machine and deep learning in the Android ecosystem. First, we presented an overview of recently released mobile chipsets that can be potentially used for accelerating the execution of neural networks on smartphones and other portable

devices, and provided an overview of the latest changes in the Android machine learning pipeline. We described the changes introduced in the current AI Benchmark release and discussed the results of the floating-point and quantized inference obtained from the chipsets produced by Qualcomm, HiSilicon, Samsung, MediaTek and Unisoc that are providing hardware acceleration for AI inference. We compared the obtained numbers to the results of desktop CPUs and GPUs to understand the relation between these hardware platforms. Finally, we discussed future perspectives of software and hardware development related to this area and gave our recommendations regarding the deployment of deep learning models on smartphones.

References

- [1] Android Neural Networks API 1.2. <https://android-developers.googleblog.com/2019/03/introducing-android-q-beta.html>. 2
- [2] Martín Abadi, Paul Barham, Jianmin Chen, Zhifeng Chen, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Geoffrey Irving, Michael Isard, et al. Tensorflow: a system for large-scale machine learning. In *OSDI*, volume 16, pages 265–283, 2016. 2, 11, 12
- [3] Hassan Abu Alhaja, Siva Karthik Mustikovela, Lars Mescheder, Andreas Geiger, and Carsten Rother. Augmented reality meets deep learning for car instance segmentation in urban scenes. In *British machine vision conference*, volume 1, page 2, 2017. 1
- [4] Eric Anquetil and Hélène Bouchereau. Integration of an on-line handwriting recognition system in a smart phone device. In *Object recognition supported by user interaction for service robots*, volume 3, pages 192–195. IEEE, 2002. 1
- [5] Android Neural Networks API. <https://developer.android.com/ndk/guides/neuralnetworks>. 2
- [6] ArmNN. <https://github.com/arm-software/armnn>. 4
- [7] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv preprint arXiv:1409.0473*, 2014. 1
- [8] Sharan Chetlur, Cliff Woolley, Philippe Vandermersch, Jonathan Cohen, John Tran, Bryan Catanzaro, and Evan Shelhamer. cudnn: Efficient primitives for deep learning. *arXiv preprint arXiv:1410.0759*, 2014. 2, 12
- [9] Dan Claudiu Ciresan, Ueli Meier, Jonathan Masci, Luca Maria Gambardella, and Jürgen Schmidhuber. Flexible, high performance convolutional neural networks for image classification. In *Twenty-Second International Joint Conference on Artificial Intelligence*, 2011. 3
- [10] NVIDIA CUDA. <https://developer.nvidia.com/cuda-zone>. 2, 12
- [11] TensorFlow Lite GPU delegate. <https://www.tensorflow.org/lite/performance/gpu>. 7
- [12] TensorFlow Lite delegates. <https://www.tensorflow.org/lite/performance/delegates>. 2, 7
- [13] PyTorch Android Demo. <https://github.com/cedrickchee/pytorch-android>. 6
- [14] PyTorch AI Camera Demo. <https://github.com/caffe2/aicamera>. 6
- [15] PyTorch Neural Style Transfer Demo. <https://github.com/caffe2/aicamera-style-transfer>. 6
- [16] Daniel DeTone, Tomasz Malisiewicz, and Andrew Rabinovich. Deep image homography estimation. *arXiv preprint arXiv:1606.03798*, 2016. 1
- [17] Chao Dong, Chen Change Loy, Kaiming He, and Xiaoou Tang. Image super-resolution using deep convolutional networks. *IEEE transactions on pattern analysis and machine intelligence*, 38(2):295–307, 2016. 1, 10, 11
- [18] COIN Emmett, Deborah Dahl, and Richard Mandelbaum. Voice activated virtual assistant, Jan. 31 2013. US Patent App. 13/555,232. 1
- [19] AI Benchmark: Ranking Snapshot from September 2018. <https://web.archive.org/web/20181005023555/ai-benchmark.com/ranking>. 2
- [20] Abdenour Hadid, JY Heikkila, Olli Silvén, and M Pietikainen. Face and eye detection for person authentication in mobile phones. In *2007 First ACM/IEEE International Conference on Distributed Smart Cameras*, pages 101–108. IEEE, 2007. 1
- [21] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. In *European conference on computer vision*, pages 630–645. Springer, 2016. 11
- [22] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997. 3, 10, 11
- [23] Andrew G Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*, 2017. 1
- [24] Baotian Hu, Zhengdong Lu, Hang Li, and Qingcai Chen. Convolutional neural network architectures for matching natural language sentences. In *Advances in neural information processing systems*, pages 2042–2050, 2014. 1
- [25] Jonathan Huang, Vivek Rathod, Chen Sun, Menglong Zhu, Anoop Korattikara, Alireza Fathi, Ian Fischer, Zbigniew Wojna, Yang Song, Sergio Guadarrama, et al. Speed/accuracy trade-offs for modern convolutional object detectors. In *IEEE CVPR*, volume 4, 2017. 1
- [26] Andrey Ignatov. Real-time human activity recognition from accelerometer data using convolutional neural networks. *Applied Soft Computing*, 62:915–922, 2018. 1
- [27] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. Dslr-quality photos on mobile devices with deep convolutional networks. In *the IEEE Int. Conf. on Computer Vision (ICCV)*, 2017. 1, 10, 11
- [28] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. Wespe: weakly supervised photo enhancer for digital cameras. *arXiv preprint arXiv:1709.01118*, 2017. 1
- [29] Andrey Ignatov, Nikolay Kobyshev, Radu Timofte, Kenneth Vanhoey, and Luc Van Gool. Wespe: weakly supervised photo enhancer for digital cameras. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 691–700, 2018. 10, 11

- [30] Andrey Ignatov and Radu Timofte. Ntire 2019 challenge on image enhancement: Methods and results. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition Workshops*, pages 0–0, 2019. **1**
- [31] Andrey Ignatov, Radu Timofte, William Chou, Ke Wang, Max Wu, Tim Hartley, and Luc Van Gool. Ai benchmark: Running deep neural networks on android smartphones. In *Proceedings of the European Conference on Computer Vision (ECCV)*, pages 0–0, 2018. **2, 4, 5, 6, 7, 9, 13, 15**
- [32] Andrey Ignatov, Radu Timofte, et al. Pirm challenge on perceptual image enhancement on smartphones: Report. In *European Conference on Computer Vision Workshops*, 2018. **1**
- [33] Dmitry Ignatov and Andrey Ignatov. Decision stream: Cultivating deep decision trees. In *2017 IEEE 29th International Conference on Tools with Artificial Intelligence (IC-TAI)*, pages 905–912. IEEE, 2017. **1**
- [34] Benoit Jacob, Skirmantas Kligys, Bo Chen, Menglong Zhu, Matthew Tang, Andrew Howard, Hartwig Adam, and Dmitry Kalenichenko. Quantization and training of neural networks for efficient integer-arithmetic-only inference. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2704–2713, 2018. **8**
- [35] Jiwon Kim, Jung Kwon Lee, and Kyoung Mu Lee. Accurate image super-resolution using very deep convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1646–1654, 2016. **10, 11**
- [36] Masashi Koga, Ryuji Mine, Tatsuya Kameyama, Toshikazu Takahashi, Masahiro Yamazaki, and Teruyuki Yamaguchi. Camera-based kanji ocr for mobile-phones: Practical issues. In *Eighth International Conference on Document Analysis and Recognition (ICDAR’05)*, pages 635–639. IEEE, 2005. **1**
- [37] Raghuraman Krishnamoorthi. Quantizing deep convolutional networks for efficient inference: A whitepaper. *arXiv preprint arXiv:1806.08342*, 2018. **8**
- [38] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012. **1, 3**
- [39] Jennifer R Kwapisz, Gary M Weiss, and Samuel A Moore. Activity recognition using cell phone accelerometers. *ACM SigKDD Explorations Newsletter*, 12(2):74–82, 2011. **1**
- [40] Yann LeCun, Bernhard Boser, John S Denker, Donnie Henderson, Richard E Howard, Wayne Hubbard, and Lawrence D Jackel. Backpropagation applied to handwritten zip code recognition. *Neural computation*, 1(4):541–551, 1989. **3**
- [41] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998. **3**
- [42] Christian Ledig, Lucas Theis, Ferenc Huszár, Jose Caballero, Andrew Cunningham, Alejandro Acosta, Andrew P Aitken, Alykhan Tejani, Johannes Totz, Zehan Wang, et al. Photo-realistic single image super-resolution using a generative adversarial network. In *CVPR*, volume 2, page 4, 2017. **1, 10, 11**
- [43] Juhyun Lee, Nikolay Chirkov, Ekaterina Ignasheva, Yury Pisarchyk, Mogan Shieh, Fabio Riccardi, Raman Sarokin, Andrei Kulik, and Matthias Grundmann. On-device neural net inference with mobile gpus. *arXiv preprint arXiv:1907.01989*, 2019. **2, 7, 8**
- [44] Haoxiang Li, Zhe Lin, Xiaohui Shen, Jonathan Brandt, and Gang Hua. A convolutional neural network cascade for face detection. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 5325–5334, 2015. **1**
- [45] Intel MKL-DNN Library. <https://github.com/intel/mkl-dnn>. **2, 12**
- [46] TensorFlow Lite. <https://www.tensorflow.org/lite>. **2**
- [47] Jie Liu, Abhinav Saxena, Kai Goebel, Bhaskar Saha, and Wilson Wang. An adaptive recurrent neural network for remaining useful life prediction of lithium-ion batteries. Technical report, NATIONAL AERONAUTICS AND SPACE ADMINISTRATION MOFFETT FIELD CA AMES RESEARCH , 2010. **1**
- [48] Jiayang Liu, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. *Pervasive and Mobile Computing*, 5(6):657–675, 2009. **1**
- [49] Christos Louizos, Matthias Reisser, Tijmen Blankevoort, Efstratios Gavves, and Max Welling. Relaxed quantization for discretized neural networks. *arXiv preprint arXiv:1810.01875*, 2018. **8**
- [50] Shie Mannor, Branislav Kveton, Sajid Siddiqi, and Chih-Han Yu. Machine learning for adaptive power management. *Automatic Computing*, 10(4):299–312, 2006. **1**
- [51] VTIVK Matsunaga and V Yukinori Nagano. Universal design activities for mobile phone: Raku raku phone. *Fujitsu Sci. Tech. J*, 41(1):78–85, 2005. **1**
- [52] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013. **1**
- [53] Emiliano Miluzzo, Tianyu Wang, and Andrew T Campbell. Eyephone: activating mobile phones with your eyes. In *Proceedings of the second ACM SIGCOMM workshop on Networking, systems, and applications on mobile handhelds*, pages 15–20. ACM, 2010. **1**
- [54] Markus Nagel, Mart van Baalen, Tijmen Blankevoort, and Max Welling. Data-free quantization through weight equalization and bias correction. *arXiv preprint arXiv:1906.04721*, 2019. **8**
- [55] Jawad Nagi, Frederick Ducatelle, Gianni A Di Caro, Dan Cireşan, Ueli Meier, Alessandro Giusti, Farrukh Nagi, Jürgen Schmidhuber, and Luca Maria Gambardella. Max-pooling convolutional neural networks for vision-based hand gesture recognition. In *2011 IEEE International Conference on Signal and Image Processing Applications (ICSIPA)*, pages 342–347. IEEE, 2011. **3**
- [56] Yuval Netzer, Tao Wang, Adam Coates, Alessandro Bissacco, Bo Wu, and Andrew Y Ng. Reading digits in natural images with unsupervised feature learning. In *NIPS workshop on deep learning and unsupervised feature learning*, volume 2011, page 5, 2011. **1**
- [57] Gerrit Niezen and Gerhard P Hancke. Gesture recognition as ubiquitous input for mobile phones. In *International Workshop on Devices that Alter Perception (DAP 2008)*, in conjunction with *Ubicomp*, pages 17–21. Citeseer, 2008. **1**

- [58] Using OpenCL on Mali GPUs. <https://community.arm.com/developer/tools-software/graphics/b/blog/posts/smile-to-the-camera-it-s-opencl>. 3
- [59] Aaron van den Oord, Nal Kalchbrenner, and Koray Kavukcuoglu. Pixel recurrent neural networks. *arXiv preprint arXiv:1601.06759*, 2016. 11
- [60] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors*, 16(1):115, 2016. 1
- [61] George Papandreou, Liang-Chieh Chen, Kevin P Murphy, and Alan L Yuille. Weakly-and semi-supervised learning of a deep convolutional network for semantic image segmentation. In *Proceedings of the IEEE international conference on computer vision*, pages 1742–1750, 2015. 11
- [62] Taesung Park, Ming-Yu Liu, Ting-Chun Wang, and Jun-Yan Zhu. Semantic image synthesis with spatially-adaptive normalization. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 2337–2346, 2019. 11
- [63] TensorFlow Lite plugin for using select TF ops. <https://bintray.com/google/tensorflow/tensorflow-lite-select-tf-ops>. 6
- [64] PyTorch Lite Android port. <https://github.com/cedrickchee/pytorch-lites>. 6
- [65] Rajat Raina, Anand Madhavan, and Andrew Y Ng. Large-scale deep unsupervised learning using graphics processors. In *Proceedings of the 26th annual international conference on machine learning*, pages 873–880. ACM, 2009. 3
- [66] Google Pixel 2 Press Release. <https://www.blog.google/products/pixel/pixel-visual-core-image-processing-and-machine-learning-pixel-2/>. 6
- [67] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*, pages 234–241. Springer, 2015. 10, 11
- [68] Mark Sandler, Andrew Howard, Menglong Zhu, Andrey Zhmoginov, and Liang-Chieh Chen. Mobilenetv2: Inverted residuals and linear bottlenecks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4510–4520, 2018. 9, 11
- [69] Aarti Sathyanarayana, Shafiq Joty, Luis Fernandez-Luque, Ferda Offi, Jaideep Srivastava, Ahmed Elmagarmid, Teresa Arora, and Shahrads Taheri. Sleep quality prediction from wearable data using deep learning. *JMIR mHealth and uHealth*, 4(4), 2016. 1
- [70] Florian Schroff, Dmitry Kalenichenko, and James Philbin. Facenet: A unified embedding for face recognition and clustering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 815–823, 2015. 1
- [71] Iulian V Serban, Chinnadhurai Sankar, Mathieu Germain, Saizheng Zhang, Zhouhan Lin, Sandeep Subramanian, Taesup Kim, Michael Pieper, Sarath Chandar, Nan Rosemary Ke, et al. A deep reinforcement learning chatbot. *arXiv preprint arXiv:1709.02349*, 2017. 1
- [72] Aliaksei Severyn and Alessandro Moschitti. Twitter sentiment analysis with deep convolutional neural networks. *Proceedings of the 38th International ACM SIGIR Conference on Research and Development in Information Retrieval*, pages 959–962. ACM, 2015. 1
- [73] Siddharth Sigtia and Simon Dixon. Improved music feature learning with deep neural networks. In *2014 IEEE international conference on acoustics, speech and signal processing (ICASSP)*, pages 6959–6963. IEEE, 2014. 1
- [74] Miika Silfverberg, I Scott MacKenzie, and Panu Korhonen. Predicting text entry speed on mobile phones. In *Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 9–16. ACM, 2000. 1
- [75] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014. 11
- [76] SNPE. <https://developer.qualcomm.com/docs/snpe/overview.html>. 5
- [77] Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D Manning, Andrew Ng, and Christopher Potts. Recursive deep models for semantic compositionality over a sentiment treebank. In *Proceedings of the 2013 conference on empirical methods in natural language processing*, pages 1631–1642, 2013. 1
- [78] Jinook Song, Yunkyo Cho, Jun-Seok Park, Jun-Woo Jang, Sehwon Lee, Joon-Ho Song, Jae-Gon Lee, and Inyup Kang. 7.1 an 11.5 tops/w 1024-mac butterfly structure dual-core sparsity-aware neural processing unit in 8nm flagship mobile soc. In *2019 IEEE International Solid-State Circuits Conference-(ISSCC)*, pages 130–132. IEEE, 2019. 3, 4
- [79] Android TensorFlow Support. <https://git.io/jey0w>. 2, 6
- [80] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems*, pages 3104–3112, 2014. 1
- [81] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alexander A Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. In *AAAI*, volume 4, page 12, 2017. 10, 11
- [82] Christian Szegedy, Vincent Vanhoucke, Sergey Ioffe, Jon Shlens, and Zbigniew Wojna. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2818–2826, 2016. 1, 9, 11
- [83] Radu Timofte, Shuhang Gu, Jiqing Wu, and Luc Van Gool. Ntire 2018 challenge on single image super-resolution: Methods and results. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, June 2018. 1
- [84] Felix Von Reischach, Stephan Karpischek, Florian Michelles, and Robert Adelman. Evaluation of 1d barcode scanning on mobile phones. In *2010 Internet of Things (IOT)*, pages 1–5. IEEE, 2010. 1
- [85] Neal Wadhwa, Rahul Garg, David E Jacobs, Bryan E Feldman, Nori Kanazawa, Robert Carroll, Yair Movshovitz-Attias, Jonathan T Barron, Yael Pritch, and Marc Levoy. Synthetic depth-of-field with a single-camera mobile phone. *ACM Transactions on Graphics (TOG)*, 37(4):64, 2018. 1
- [86] Avery Wang. The shazam music recognition service. *Communications of the ACM*, 49(8):44–48, 2006. 1

- [87] Yi Wu, Jongwoo Lim, and Ming-Hsuan Yang. Object tracking benchmark. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 37(9):1834–1848, 2015. 1
- [88] Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, et al. Google’s neural machine translation system: Bridging the gap between human and machine translation. *arXiv preprint arXiv:1609.08144*, 2016. 11
- [89] Qualcomm Zeroth. <https://www.qualcomm.com/news/onq/2015/10/01/qualcomm-research-brings-server-class-machine-learning-everyday-devices-making>. 3
- [90] Hengshuang Zhao, Xiaojuan Qi, Xiaoyong Shen, Jianping Shi, and Jiaya Jia. Icnnet for real-time semantic segmentation on high-resolution images. *arXiv preprint arXiv:1704.08545*, 2017. 10, 11
- [91] Hengshuang Zhao, Jianping Shi, Xiaojuan Qi, Xiaogang Wang, and Jiaya Jia. Pyramid scene parsing network. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 2881–2890, 2017. 11