An Early Resource Characterization of Deep Learning on Wearables, Smartphones and Internet-of-Things Devices

Nicholas D. Lane[‡], Sourav Bhattacharya[‡]
Petko Georgiev[†], Claudio Forlivesi[‡], Fahim Kawsar[‡]
[‡]Bell Labs, [†]University of Cambridge

ABSTRACT

Detecting and reacting to user behavior and ambient context are core elements of many emerging mobile sensing and Internet-of-Things (IoT) applications. However, extracting accurate inferences from raw sensor data is challenging within the noisy and complex environments where these systems are deployed. *Deep Learning* – is one of the most promising approaches for overcoming this challenge, and achieving more robust and reliable inference. Techniques developed within this rapidly evolving area of machine learning are now state-of-the-art for many inference tasks (such as, audio sensing and computer vision) commonly needed by IoT and wearable applications. But currently deep learning algorithms are seldom used in mobile/IoT class hardware because they often impose debilitating levels of system overhead (e.g., memory, computation and energy). Efforts to address this barrier to deep learning adoption are slowed by our lack of a systematic understanding of how these algorithms behave at inference time on resource constrained hardware. In this paper, we present the first – albeit preliminary – measurement study of common deep learning models (such as Convolutional Neural Networks and Deep Neural Networks) on representative mobile and embedded platforms. The aim of this investigation is to begin to build knowledge of the performance characteristics, resource requirements and the execution bottlenecks for deep learning models when being used to recognize categories of behavior and context. The results and insights of this study, lay an empirical foundation for the development of optimization methods and execution environments that enable deep learning to be more readily integrated into next-generation IoT, smartphones and wearable systems.

Categories and Subject Descriptors: H.1.2 [User/Machine Systems]: Human Information Processing.

General Terms: Design, Experimentation.

Keywords: Deep Learning, Internet-of-Things, Wearables

1. INTRODUCTION

Extracting user behavior and ambient context from sensor data is a key enabler for mobile and Internet-of-Things (IoT) applications. Increasingly, emerging networked appliances (e.g., [4, 3]) monitor user activities (such as, speech, occupancy, motion) to provide an improved user experience. Similarly, for wearables and phones the tracking of the user (e.g., [24]) and surrounding conditions (e.g., [25]) has long been a core building block. Even though sensor applications and systems are highly diverse, a prominent unifying element is their need to make these types of sensor inferences.

Reliably mining real-world sensor data for this type of information remains an open problem. The world is dynamic and complex; such conditions often confuse the signal processing and machine learning techniques employed for sensor inference. The most promising approach for coping with this challenge today is *deep learning* [9, 14]. Advances in this field of machine learning have transformed how many inference tasks related to IoT and mobile applications are performed (e.g., speech [17] and face [27] recognition). Exploration of deep learning for these systems is now underway (e.g., [11, 16, 19]), with promising early results.

Despite its benefits, the adoption of deep learning within IoT and mobile hardware face significant barriers due to the resource requirements of these algorithms. Demands on memory, computation and energy make it impractical for most models to directly execute on target hardware. As a result, prominent examples of deep learning seen on phones (e.g., speech recognition) are largely cloud-assisted. This introduces important negative side-effects: first, it exposes users to privacy dangers [8] as sensitive data (e.g., audio) is processed off-device by a third party; and second, the inference execution becomes coupled to fluctuating and unpredictable network quality (e.g., latency, throughput).

Enabling wide-spread device-side deep learning inference will require a range of brand-new techniques for optimized resource sensitive execution. Our existing knowledge of deep learning algorithm behavior on constrained devices is largely limited to one-off task-specific experiences (e.g., [1, 10]). Such examples only offer a proof-by-example that forms of local execution are possible, while providing a few pointers for potential directions. What is needed is the development of techniques like off-line model optimization and runtime execution environments that shape inference-time requirements to match the resources available on target wearable, mobile or embedded platforms. A cornerstone of such efforts will be a detailed understanding of how existing algorithms perform on these platforms. Furthermore, systematic observations of deep learning runtime behavior (e.g., data/control flow) will be pivotal for understanding how to best use upcoming hardware accelerators (e.g., [11]) that perform key phases of these algorithms (e.g., convolution layers).

In this paper, we present an initial measurement study designed to provide critical first-order insights in the development of embedded and mobile device support for deep learning. Specifically, in this study we present systematic profiling of the two most commonly used deep learning model architectures (viz. CNNs – Convolutional Neural Networks, and DNNs – Deep Neural Networks) as used in four existing deep models (viz. [10, 18, 23, 21]) that process audio and image sensor data. Experiments are conducted using three hardware platforms representative of those used by IoT, wearable and mobile applications (viz. [6, 2, 5]). Most

IoT-App'15, November 1 2015, Seoul, Republic of Korea.
Copyright © 2015 ACM ISBN 978-1-4503-3838-7/15/11 ...$15.00.
DOI: http://dx.doi.org/10.1145/2820975.2820980 .

prior work has focused on how these algorithms behave *as they are scaled up*, and distributed across large cloud infrastructure for the purpose of training more robust models based on large amounts of data and more complex architectures (e.g., [13]). A core contribution of this investigation is that it provides, for the first time, insights into how these algorithms behave *as they are scaled down* – and perform inference on edge devices where resources are scarce.

Although preliminary in nature, the investigation reported here provides insights into a number of key areas. First, whole-model benchmarks (see §3) are performed for all platforms to better understand what is currently feasible, and importantly how energy consumption and execution time corresponds to existing application requirements. Second, the relationship with resource usage and internal model architecture is studied (see §4) as it relates to architecture design and related algorithmic choices. Third, and finally: memory and model footprint needs (see §5) are investigated, especially within the context of hardware capabilities. We believe these results will prove valuable as a foundation for more comprehensive measurement studies of deep learning at inference-time, and in the development of techniques to address the resource overhead these algorithms impose.

2. STUDY PRELIMINARIES

We begin with a primer on the deep learning algorithms that underpin the deep models in our study; we also briefly sketch each model, and the target hardware.

Deep Learning Primer. To understand the experimental results we now present the basic concepts of DNNs and CNNs; further details can be found in [9, 14].

Deep Neural Network. Various forms of deep learning (e.g., Restricted Boltzmann Machines, Deep Belief Networks) share a common architecture shown in Figure 1, and are often collectively referred to as Deep Neural Networks. A DNN is comprised of fully-connected layers, each of which contain a collection of units (or nodes). Raw data (e.g., audio, images) initialize the values of the first layer (the input layer). The output layer (the last layer) corresponds to inference classes, with units capturing individual inference categories (e.g., music or cat). Hidden layers are contained between input and output layers. Collectively, they are responsible for transforming the state of the input layer into the inference classes captured in the last layer. Every unit contains an activation function that determines how to calculate the units's own state based on units from the immediately previous layer. The degree of influence of units between layers vary on a pairwise basis determined by a weight value.

Inference for a DNN occurs through a feed-forward algorithm that operates on sensor data segments in isolation. The algorithm begins at the input layer and moves forward layer by layer, updating the state of each unit one by one. The process terminates at the output layer when all units have been updated. The inferred class corresponds to the output layer unit with the greatest state value.

Convolutional Neural Networks. CNNs are an alternative to DNNs that still share many architectural similarities. As shown in Figure 2, a CNN is composed of one or more: convolutional layers, pooling or sub-sampling layers, and fully connected layers (with this final type being equivalent to those used in DNNs). The aim of these layers is to extract simple representations at high resolution from

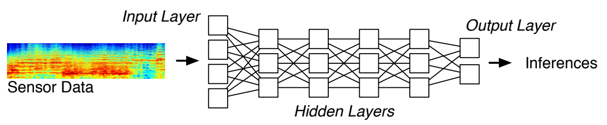


Figure 1: A DNN contains fully-connected feed-forward layers.

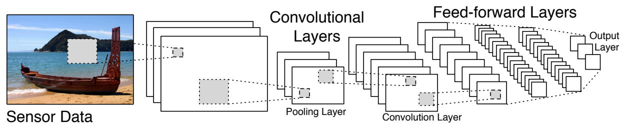


Figure 2: A CNN mixes convolutional and feed-forward layers.

the input data, and then converting these into more complex representations, but at much coarser resolutions within subsequent layers. This is achieved by first applying convolutional filters (with small kernel width) to capture local data properties. Next follow max or min pooling layers causing representations to be invariant to translations, this also acts as a form of dimensionality reduction. Finally, fully connected layers (i.e., a DNN) are applied to complete classification.

Inference under a CNN is very similar to that of a DNN. Again, inference operates only on a single segment of data at a time. Sensor data is first vectorized into two dimensions. Next, data is provided to convolutional layers at the head of the architecture. This can be considered a form of feature extraction before the fully connected layers are engaged. Inference then proceeds exactly as previously described for DNNs until ultimately a classification is reached.

	Type	**Size**	**Architecture**
AlexNet	CNN	60.9M	c:5[ı]; p:3[‡]; h:2[⋆]; n:{all 4096}[†]
SVHN	CNN	313K	c:2[ı]; p:2[‡]; h:2[⋆]; n:{1600,128}[†]
Deep KWS	DNN	241K	h:3[⋆]; n:{all 128}[†]
DeepEar	DNN	2.3M	h:3[⋆]; n:{all 512 or 256}[†]

[ı]*convolution layers*; [‡]*pooling layers*; [⋆]*hidden layers*; [†]*hidden nodes*

Table 1: Deep Learning Models

Representative Deep Models. Table 1 presents key characteristics of the models examined in this study.

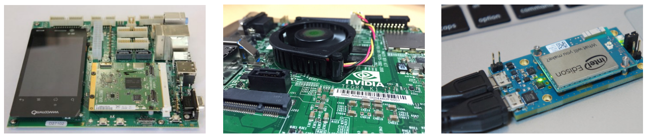
AlexNet. This object recognition model [18] supports more than 1,000 object classes (e.g., dog, car), and is by far the most complex model studied (60.9M parameters, the next highest is just 2.3M). In 2012, it offered state-of-the-art levels of accuracy for well-known datasets like ImageNet.

SVHN. Specializing in extracting numbers from complex and noisy scenes, this model [23] has been used to recognize house numbers from Google Streetview images.

Deep KWS. Designed for resource constrained devices, this audio model [10] recognizes 22 spoken words. It targets use cases like phones reacting to specific phrases ("Hey Siri"), an inference task often called keyword spotting (KWS).

DeepEar. This also is an audio model [21] designed for constrained use. Unusually, it is a composite of 3 coupled DNNs offering separate recognition tasks (viz. emotion recognition, speaker identification and ambient sound classification).

Target Hardware Platforms. Experiments are performed on three hardware platforms detailed below. Figure 3 shows breakout boards used by each SoC.



(a) Snapdragon 800 (b) Tegra K1 (c) Edison

Figure 3: Wearable, Smartphone and IoT Hardware

Qualcomm Snapdragon 800. By a wide margin, this SoC [6] is the most widely available of the three platforms and is shipping, for example, inside a variety of smartphones (e.g., Nexus 5). Primarily designed for phones and tablets, it contains 3 processors: a Krait 4-core 2.3 GHz CPU, an Adreno 330 GPU and a 680 MHz Hexagon DSP. We find the CPU can address 1GB of RAM, but the DSP only 8MB.

Intel Edison. Targeting wearables and form-factor sensitive IoT, the Edison [2] is the smallest (3.5 x 2.5 x 0.39 cm) but least computational powerful of all tested hardware. However, it still has a 500MHz dual-core Atom "Silvermont" CPU assisted by a 100 MHz Quark processor. Perhaps surprisingly given its size, it also includes 1 GB of RAM.

Nvidia Tegra K1. This SoC [5] provides extreme GPU performance not found in other mobile and IoT hardware. The heart of this chip is the Kepler 192-core GPU, which is coupled with a 2.3 GHz 4-core Cortex CPU and an additional low-power *5th* core (LPC) that is designed for energy efficiency. The K1 SoC is used in IoT devices such as June IoT Oven [3] and IoT-enabled cars. Mobile examples of the Tegra include the Nexus 9 along with the development smartphone in Google's Project Tango. The CPU can access up to 1.7GB of RAM, largest of all processors profiled.

3. PERFORMANCE BENCHMARKS

Our first set of experiments are designed to test the raw feasibility of executing deep learning models across our target hardware plaforms. We also consider end-to-end model performance metrics of execution time and energy consumption, especially in terms of how these map to application needs.

Experiment Setup. For these experiments, we measure the execution time and energy consumption of each hardware platform using a Monsoon Power Monitor and code annotation; every model performs 1,000 separate inferences, we report average performance levels. Datasets (and input specifications such as image and audio parameters) for inference and model training closely adhere to the expectations of the model authors/developers. We use two implementations for our experiments. Tegra based profiling uses Torch [7], a deep learning framework that includes a number of state-of-the-art optimizations; we also use Torch for training all models used in all experiments. Snapdragon and Edison experiments instead use a C/C++ implementation of DNN and CNN inference that we internally develop. This implementation is able to load Torch trained models. In cases where the platform includes multiple processors we perform individual tests on each one available for use. Note, the Snapdragon GPU is not currently supported by our implementation and the Edison Quark is not openly programmable at the time of our tests; consequently, neither is used in experiments.

We also perform coarse estimates of battery life. As all deep models selected use either image or audio data, we use energy usage for mobile microphones and cameras reported in [22, 15]. For the processor-dependent cost of sensor interaction, we assume a constant compute time across all platforms based on the actual time observed within the Snapdragon. Otherwise processors are assumed to only perform model inference. A 5 second pause between inferences occurs, some processors (Snapdragon and Edison) are able to engage a low-power idle mode during this waiting period. Finally, estimates assume the processor is powered from a 2000mAH battery with a perfect discharge curve.

	Tegra		Snapdragon		Edison
	CPU	GPU	CPU	DSP	CPU
Deep KWS	2.2	5.2	11.3	10.2	12.1
DeepEar	18.7	15.2	119.1	19.9	21.3
AlexNet	1,678.6	232.2	256,925.3	-	110,385.3
SVHN	43.1	13.3	2,604.9	-	1,389.2

Table 2: Energy Consumption (mJ.)

	Tegra		Snapdragon		Edison
	CPU	GPU	CPU	DSP	CPU
Deep KWS	0.8	1.1	7.1	7.0	63.1
DeepEar	6.7	3.2	71.2	379.2	109.0
AlexNet	600.2	49.1	159,383.1	-	283,038.6
SVHN	15.1	2.8	1,616.5	-	3,562.3

Table 3: Execution Time (msec.)

Feasibility Observations. Table 2 and 3 provide a performance snapshot for all studied deep models running on the target mobile- and IoT-class hardware. Almost all model and processor combinations (18 out of 20[1]) are able to execute; even large-scale models like AlexNet are supported by the weakest of our processors (Edison). This indicates a deep learning approach to a wide variety of inference tasks (the 6 different tasks described in §2) is possible on latest versions of constrained hardware. Furthermore, because the architecture (e.g., layer types, layer and node size) is the primary factor in determining if a deep model will execute on a specific processor; then, this result also suggests models of similar architecture to our study set – but targeting different inference tasks – will also function to some degree. The range of inference tasks offered by this set of similar models is enormous [9, 14], and comprise tasks generally not seen on IoT or mobile/wearable hardware; examples include: place classification (categories like: library, gas station, art gallery), gender and age estimation, textual descriptors of images and sounds, and language understanding. This is significant because deep models have been ignored for years due to concerns over their basic feasibility on wearable and IoT hardware. Although, it is important to keep in mind the extreme performance issues that still remain to be solved (described next). Moreover, many deep models remain out of reach due to their shear complexity; while AlexNet (60.9M parameters) barely executes on study hardware, the latest deep methods for important tasks are much larger, two examples being DeepFace [27] (120M) used for face recognition and VGG [26] (143M) that recognizes objects.

Energy and Latency. Beyond indicating basic levels of feasibility, Table 2 and 3 also highlight clear examples of serious performance bottlenecks. As would be expected, simpler models (as measured by the number of model parameters) such as Deep KWS have acceptable performance (with execution times ranging between 63 and <1 msec. across all

[1]AlexNet and SVHN can not execute on the Snapdragon DSP because it is limited to only 8MB of RAM.

hardware). DeepEar is an exception to this tendency; for instance, although it has 7× more parameters than SVHN it is also on average is 22× *faster* in execution time. This follows the pattern of CNN performance being much worse than their DNN counterparts. We observe the non-GPU processors appear to struggle with convolutional layers, this is most conspicuous for AlexNet under the Snapdragon and Edison where it has execution times of 2.6 and 4.7 *minutes* respectively. To put these values in context, audio sensing with shallow models (such as, Gaussian Mixture Models and features like Perceptual Linear Prediction) have been shown to require ≈ 500 msec. for emotion recognition on the Snapdragon CPU [15], and 105 msec. for speaker identification on the Edison [20]. Collectively, these findings emphasize the ease by which deep models can overwhelm target hardware. We observe layer type appears to have a large bearing on energy and execution efficiency. There is also evidence of the efficiency of executing certain layer types varying widely depending on the processor type; for example, the Tegra GPU is by far the most efficient processor for CNN layers while DNN layers have a more uniform processor performance.

	Tegra		Snapdragon		Edison
	CPU	GPU	CPU	DSP	CPU
Deep KWS	14.34	9.16	5.00	134.41	27.78
DeepEar	21.74	22.02	16.93	342.47	30.99
AlexNet	3.49	10.36	3.80	-	13.88
SVHN	13.98	14.81	3.97	-	15.38

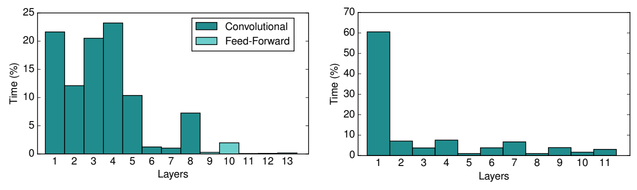
Table 4: Battery Life Estimate (hrs.)

Application Requirements. Execution time for deep models is an important aspect as it dictates the responsiveness of an application to events and user actions. Table 3 shows inference is often completed in < 1 sec. Superficially, this suggests execution time is not a bottleneck. However, the table also shows model complexity at the AlexNet level causes latencies of multiple minutes on wearable hardware. This is not adequate for systems needing near real-time responses like fall detection or cognitive support. Nevertheless, in contrast to response time sensitive systems, all models are suitable for life-logging applications that tolerate long processing delays (as the aim is to collect longitudinal data).

Energy consumption is an equally important metric. Many applications run continuously and need to track user behavior and context all day (e.g., [24]), ideally these systems must have a battery life that spans the waking hours of users (e.g., 16 to 18 hours). From Table 4, we see only the DeepEar DNN models running on the various processors and the Deep KWS running on the Snapdragon DSP or Edison result in battery lifetimes of this length. AlexNet, and to a lesser extent SVHN, cause poor battery life due to the compute time they require. For instance, on the Snapdragon CPU when these CNNs are continuously processing image data a standard 2000mAh battery would last < 4 hours. In contrast, the Tegra executes all models very quickly and so maintains a very low per-inference cost; but critically it lacks the ability to idle in a low-power state and this severely limits battery life. Excluding the issue of execution delays, our analysis also indicates broad support for a delay-tolerant life-logging applications as tens or hundreds of inferences can be performed with only a fraction of the battery. For example, for less than 20% of the battery, we find even AlexNet can execute > 4000× on the Tegra CPU, > 16× on the Snapdragon CPU, and > 25× on the Edison CPU.

	Layer type	**Tunable parameters**	**Time (%)**
1	Convolution	34, 944	37.20
2	Non-linear	-	0.05
3	Normalization	-	0.12
4	Pooling	-	0.15
5	Convolution	307,456	2.05
6	Non-linear	-	0.05
7	Normalization	-	0.21
8	Pooling	-	1.11
9	Convolution	885,120	30.89
10	Non-linear	-	0.46
11	Convolution	663,936	13.56
12	Non-linear	-	0.08
13	Convolution	442,624	7.45
14	Non-linear	-	0.38
15	Pooling	-	0.74
16	Feed-forward	37,752,832	0.49
17	Non-linear	-	0.15
18	Dropout	-	0.06
19	Feed-forward	16,781,312	0.19
20	Non-linear	-	0.14
21	Dropout	-	0.07
22	Feed-forward	4,097,000	4.34
22	Softmax	-	0.06

Table 5: Layer-by-layer runtime performance of AlexNet. Average inference time for a color image of dimension $3 \times 256 \times 256$ is 67 msec. approximately. The layers can be conceptually grouped into two cases: convolution-oriented feature extraction and feed-forward based classification; this division is shown with the horizontal line between layers 15 and 16. (Execution on Tegra GPU).



(a) SVHN **(b)** Deep KWS

Figure 4: Layer-by-layer computational overhead for one DNN and one CNN. (Execution on Tegra GPU).

4. LAYER AND UNIT PROFILING

We next examine factors contributing to the computation-related overhead observed in the prior section. Computational load is tied closely with other performance metrics including execution time and energy efficiency. We focus in particular on the consequences of architecture (e.g., layer type) and algorithmic choices (e.g., activation function).

Experiment Setup. We repeat the same core experiments described in §3, however through code annotation the execution time of individual layers and units is profiled. We highlight performance characteristics not tied to platform differences, and emphasize this by reporting the percentage of execution time (for unit, or layer) as the primary metric used. Similarly, all reported effects are based on measurements from a single platform – the Tegra GPU; importantly, we also observe the same performance trends across other test hardware platforms. Experiments to test the sensitivity to activation functions are done by replacing the relevant layer, retraining the model, and repeating inference tests.

Layer Analysis. Table 5 shows each internal layer of the CNN-based AlexNet model. For each layer we provide the number of parameters and the relative contribution of the layer to overall model execution time. Clearly compute

	Layer type	Tunable parameters	Time (%)
1	Feed-forward	115,200	22.30
2	Non-linear	-	1.40
3	Dropout	-	2.0
4	Feed-forward	262,656	32.60
5	Non-linear	-	1.30
6	Dropout	-	1.50
7	Feed-forward	262,656	32.40
8	Non-linear	-	1.10
9	Dropout	-	1.20
10	Feed-forward	7,182	3.20
11	Softmax	-	1.10

Table 6: Layer-by-layer performance of the DNN within DeepEar that performs emotion recognition. Inference time for a 30 msec. audio frame is 0.00035 msec. (Execution on Tegra GPU).

	Sigmoid (%)	Tanh (%)	ReLU (%)
Deep KWS	6.3	0.4	0.2
DeepEar *(emotion only)*	6.1	0.4	0.2

Table 7: Contribution to overall runtime of non-linear layers for Deep KWS and DeepEar (only the DNN responsible for emotion recognition). Influence of activation function choice is shown. (Execution on Tegra GPU).

time varies widely between layers, with a general trend of early layers requiring more computation than later ones – an effect largely due to the dimensionality reduction occurring as progressive pool layers take effect. This trend is even more strongly present in SVHN (Figure 4a) that, like AlexNet, is CNN-based. Table 5 also shows a pronounced division between the time taken by layers tied to the convolutional operations (94.5%) and the later feed-forward layers (5.5%). Interestingly however, these same convolution tied layers only account for 2.3M (3.7%) of the 60.9M parameters of this model; this has important implications for managing model footprint discussed further in §5. But, with respect to shaping computational usage, this is a strong signal that performance for CNNs can be improved by leaning on additional feed-forward layers as substitutes for convolutional ones (where possible given an inference task). This is acutely relevant for platforms that are computation-limited but with adequate memory to cope with the additional parameters that come with extra feed-forward layers; the wearable-class Edison, for example, precisely matches this hardware profile.

In contrast, Table 6 presents the same analysis but for the DNN-based DeepEar (specifically, we show the internal DNN responsible for emotion recognition). A flatter relationship between layer depth and computational load is seen in this table than in two prior CNN models, with computation even increasing during middle layers – a pattern not seen in any other deep model we study. Although, Deep KWS (Figure 4b) reverts back to the same pattern established by the CNNs and in fact has the steepest decline of per-layer computation across all models. Collectively, these results highlight the uniqueness of the internal computational load profile each model brings to bear on hardware.

Unit Analysis. Common design choices at the unit-level largely relate to the activation function and the use of techniques like dropout [12] that influence how nodes interconnect. Table 7 shows the impact of selecting one of three popular activation functions, in terms of how their computation throughout the model contributes to overall execution

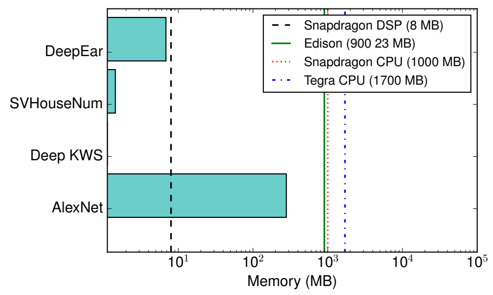


Figure 5: Memory required to complete end-to-end inference, the whole model resides in memory during this process. The maximum availabile memory to the test processors are shown for comparison purposes. (Execution on Snapdragon CPU).

time for the two DNN-based models. This table indicates none of these choices have the same level of import as the decision to use a convolution or feed-forward layer (that routinely consume more than 25% each). However, sigmoid is clearly the most expensive – although fortunately its usage is declining in favor of functions like ReLU that are faster to compute. Similarly, dropout is both increasing in usage (appearing, for example in AlexNet and DeepEar) and has significantly lower computation – in this case it causes 50% of pairwise unit connections in both models to be ignored and so do not need to be calculated (this depends on the dropout parameter used, though 0.5 is common). Finally, the parameters of convolutions in early phases of CNNs also can have large effects on runtime. We find the runtime of these layers are dominated by the size and number of filters.

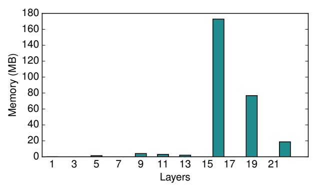
5. MODEL FOOTPRINT

Our final experiments examine the memory footprint of each model, this is known to be a key bottleneck in the use of deep learning on resource constrained devices due to their many parameters. We compare model memory requirements against availability on each hardware platform.

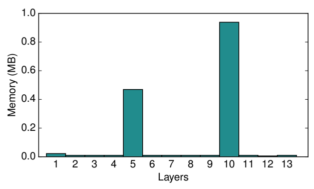
Experiment Setup. In these experiments we determine overall, and per layer, memory usage during inference for each model using the aforementioned C/C++ implementation. An aim of this implementation is to carefully use memory during execution, and in its representation of models. By default the whole model resides in memory during processing. However, for per-layer profiling this behavior is changed and only the layers being operated upon reside in memory. We report measurements from the Snapdragon CPU, we find these generalize well across the platforms.

Whole Model Requirements. Figure 5 shows the overall DNN/CNN model size across all layers in comparison to the maximum available memory. From the figure we can see this is a potential bottleneck especially in the case of mobile device memory that will be typically shared by multiple applications. For instance, AlexNet occupies nearly 300 MB which is roughly a third of the memory available for the Edison and Snapdragon CPU, and is 37× the maximum memory available to the Snapdragon DSP. This problem is even worse for previously mentioned (see §3) models like DeepFace (462MB) and VGG (548MB) that have even larger memory requirements[2] Two implications from these results are: first, even when processors have sufficient computa-

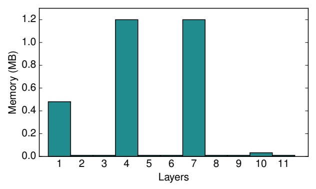
[2]Memory for these two models are estimated, and not measured.



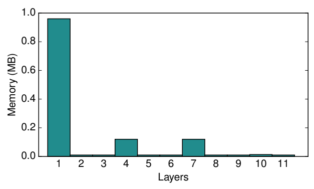
(a) AlexNet



(b) SVHN



(c) DeepEar *(emotion only)*



(d) Deep KWS

Figure 6: Memory requirements during inference on a per layer basis; only the layers of the model being operated upon are left in memory to lower requirements. (Execution on Snapdragon CPU).

tional resources to run these models they maybe prevented due to memory bottlenecks; and second, this promotes solutions that partition inference *across* processors (when their memory varies significantly, as the case in experiments).

Memory Fluctuations. While Figure 5 shows the overall memory requirement for a complete inference to be performed, Figure 6 displays per-layer memory requirements. As shown, the memory needs of deep models can fluctuate dramatically from one layer to another, and may only peak for a short burst of time. For example, Figure 6a shows 93% of memory is only required during the processing of 3 of the 22 layers, with all 3 of the layers occurring briefly in the later part of inference. This is an important finding because memory usage under deep models is much more bursty and transient than expected. Thus, mitigating techniques need only focus on a few layers; this also suggests running concurrent deep models might be more feasible than first thought.

Figure 6 also illustrates the considerable difference between the memory needs of DNNs and CNNs. CNNs require far less space than their DNN counterparts. This is due to convolution layers requiring far fewer parameters than those of DNNs (although of course they still require a lot of computation). Similar to our prior comments (see §4) regarding computation, this implies for platforms that are restricted in memory, but have computation available, then the use of feed-forward layers might be reduced. One example of this is the Snapdragon DSP, which is severely memory limited but has a processor that excels at certain computations. Interestingly, this intersects with ongoing discussions inside the computer vision community as to the value of fully connected layers that follow convolutional ones. It is suggested performance can improve if feed-forward layers (in CNNs) are replaced with cheaper to compute shallow methods.

6. CONCLUSION

In this work, we perform a preliminary measurement study into deep learning algorithms relevant to IoT and mobile applications. The aim of this study has been two fold. First, to provide initial observations as to the execution performance of this type of sensor processing when deployed on mobile- and IoT-class hardware. Second, to begin building the knowledge necessary to design general-purpose solutions for both reducing and managing the resources consumed by these learning algorithms. Few studies of this type currently exist, we hope these early findings prove valuable to those researchers seeking to bring deep learning into more widespread usage within IoT and wearable systems.

7. REFERENCES

[1] How Google Translate Squeezes Deep Learning onto a Phone. `http://googleresearch.blogspot.co.uk/2015/07/how-google-translate-squeezes-deep.html`.

[2] Intel Edison. `http://www.intel.com/content/www/us/en/do-it-yourself/edison.html`.

[3] June Oven. `http://juneoven.com/`.

[4] Nest Themostat. `http://nest.com/thermostat/meet-nest-thermostat`.

[5] Nvidia Tegra K1. `http://www.nvidia.com/object/tegra-k1-processor.html`.

[6] Qualcomm Snapdragon 800. `http://www.qualcomm.com/products/snapdragon/processors/800`

[7] Torch. `http://torch.ch/`.

[8] Your Samsung SmartTV Is Spying on You, Basically. `http://www.thedailybeast.com/articles/2015/02/05/your-samsung-smarttv-is-spying-on-you-basically.html`.

[9] Y. Bengio, et al. Deep Learning. MIT Press, 2015.

[10] G. Chen, et al. Small-footprint Keyword Spotting using Deep Neural Networks. *ICASSP '14*.

[11] T. Chen, et al. Diannao: A Small-footprint High-throughput Accelerator for Ubiquitous Machine-learning. *ASPLOS '14*.

[12] G. E. Dahl, et al. Improving Deep Neural Networks for LVCSR using Rectified Linear Units and Dropout. *ICASSP '13*.

[13] J. Dean, et al. Large Scale Distributed Deep Networks. *NIPS '12*.

[14] L. Deng and D. Yu. Deep Learning: Methods and Applications. Now Publishers, 2014.

[15] P. Georgiev, et al. DSP.Ear: Leveraging Co-processor Support for Continuous Audio Sensing on Smartphones. *SenSys '14*.

[16] N. Hammerla, et al. PD Disease State Assessment in Naturalistic Environments using Deep Learning. *AAAI '15*.

[17] G. Hinton, et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition. *Signal Processing Magazine*, 2012.

[18] A. Krizhevsky, et al. Imagenet Classification with Deep Convolutional Neural Networks. *NIPS '12*.

[19] N. D. Lane, et al. Can Deep Learning Revolutionize Mobile Sensing? *HotMobile '15*.

[20] N. D. Lane, et al. Zoe: A Cloud-less Dialog-enabled Continuous Sensing Wearable Exploiting Heterogeneous Computation. *MobiSys '15*.

[21] N. D. Lane, et al. Deepear: Robust Smartphone Audio Sensing in Unconstrained Acoustic Environments using Deep Learning. *UbiComp '15*.

[22] R. LiKamWa, et al. Energy Characterization and Optimization of Image Sensing Toward Continuous Mobile Vision. *MobiSys '13*.

[23] Y. Netzer, et al. Reading Digits in Natural Images with Unsupervised Feature Learning. *NIPS workshop on deep learning and unsupervised feature learning*. 2011.

[24] M. Rabbi, et al. Passive and In-situ Assessment of Mental and Physical Well-being using Mobile Sensors. *UbiComp '11*.

[25] S. Rallapalli, et al. Enabling Physical Analytics in Retail Stores using Smart Glasses. *MobiCom '14*.

[26] K. Simonyan, et al. Very Deep Convolutional Networks for Large-scale Image Recognition. *ICLR '15*.

[27] Y. Taigman, et al. Deepface: Closing the Gap to Human-level Performance in Face Verification. *CVPR '14*.