CNN-Based Camera-less User Attention Detection for Smartphone Power Management

CNN-Based Camera-less User Attention Detection for Smartphone Power Management

Daniele Jahier Pagliari, Matteo Ansaldi, Enrico Macii and Massimo Poncino
Politecnico di Torino
Turin, Italy
name.first_surname@polito.it

***Abstract*—The many sensors hosted by mobile electronic devices are commonly used to recognize user activities and context, in order to provide new functionalities, such as tracking physical activity and sleep cycles. Despite its potential, such context recognition is only employed for power management purposes in very specific scenarios (e.g. in-pocket detection).**

In this work we present a novel context recognition system able to reliably identify whether a mobile device is not being looked at, and to consequently trigger power management actions such as turning off the display and moving to suspended mode. Our method takes as input the readings from common low-power sensors present in virtually all mobile devices and classifies them using a Convolutional Neural Network. Most importantly, the power-hungry camera sub-system is not used, resulting in an extremely energy-efficient detection strategy.

Results show that our system is able to identify scenarios in which a device is not being used with 95.6% accuracy, thus reducing the energy overheads by 91% compared to a standard timeout-based power management and by 58% compared to a system relying on the camera.

***Index Terms*—Power Management; Neural Networks;**

I. Introduction

The possibility of detecting the user context in sensor-rich mobile devices has been used to provide novel functionalities such as activity recognition, monitoring of elderly patients, etc. [1]. However, context detection has only seldom been used for power management, and mostly for simple tasks such as turning off the screen when a smartphone is inside a pocket or during a phone call [2]. Nonetheless, context information provides very valuable knobs for optimally tuning the power/performance tradeoff of these devices, e.g., by immediately turning off the display when the phone is held in one's hand during a dynamic activity such as running, or reducing the quality of audio/video playback based on the noise and light conditions.

Among the several context detection methods studied in literature, supervised learning based on neural networks has been shown to provide the best quality results; moreover, it has also been shown that these models can be affordably executed on mobile devices, despite their complexity [3], [5], [6]. In fact, the most computationally complex task involved with neural networks, i.e. their training, can be performed offline in the cloud using high-end GPUs. The mobile device should only perform the final classification (inference) of inputs using a pre-trained model, a task that can be easily handled by high-performance mobile devices.

In this work we consider the use of context information to improve the result of smartphones/tablets power management. Specifically, we propose a new system able to automatically detect if a device is being looked at or not, and consequently trigger power management decisions, such as turning off the display or transitioning to suspended state.

Our system is based on a Convolutional Neural Network (CNN) classifier that takes as input the data coming from common low-power sensors, i.e. accelerometer, ambient light, proximity and touch. The key feature of our method is to avoid the use of energy-hungry cameras, which results in significant reduction of the energy overheads and, as a side effect, also eliminates possible security concerns. Trained on data gathered from a pool of volunteers, our system achieves a 96% accuracy in detecting those scenarios that require a power management action. Moreover, the energy overheads when the device is unused are 58% lower than those obtained using the camera, and 91% lower than those linked with a standard timeout-based power management.

II. Background and Related Work

There have been multiple attempts to use the array of sensors available on today's smartphones and tablets for activity and context detection. The authors of [2] perform a thorough review of the literature focusing on smartphone resource and energy management by means of context detection. Their survey clearly shows how most solutions focus only on spatial (location) and temporal (time of the day/year) context, using such information to disable or manage specific power-hungry peripherals, such as the cellular interface, WiFi and GPS. Examples include scanning for Wi-Fi more or less frequently depending on location and based on the history of previous connections, or tuning the frequency of GPS sensing depending on the state of charge of the battery.

Other works perform more advanced forms of context and activity detection, but do not exploit them for energy management. The authors of [7] propose a context-detection system for Android able to detect a user's location and activity based on radio signals (GPS, UMTS, WLAN), camera for light detection, microphone, accelerometer and compass as inputs. Classification is performed using a neural network. In [1], wearable inertial sensors are used to recognize human activities using multiple shallow supervised and unsupervised learning techniques. Data is collected using three inertial

units placed on the chest, right thigh and left ankle of the participants. Twelve activities are classified including standing, sitting, walking, etc. Classification is performed both using raw data and extracted time/frequency domain features.

The authors of [3] analyse and compare multiple sensor fusion approaches using both shallow and deep learning classifiers for context detection. Vanilla Deep Neural Networks (DNN) and Convolutional Neural Networks (CNN) are compared against Random Forests (RF) and Decision Trees (DT). Furthermore, deep learning approaches are tested with two different architectures; *feature concatenation*, where a single NN receives all sensors inputs, and *modality specific*, where each type of input is first pre-processed by dedicated network layers, whose outputs are then concatenated and fed to a single classifier. These approaches are compared on four different activity datasets showing that deep classifiers yield a 27% higher accuracy on average. Specifically, CNNs outperform all other methods in most datasets, and modality specific solutions are generally better than feature concatenation. An even more recent work proposes Convolutional Long-Short Time Memory (LSTM) networks for gesture detection [4].

Due to the detailed comparative analysis performed in [3] on multiple datasets, we have used their results as an inspiration for constructing the classifier required by our method. However, the goal of our system is completely different; while they aim at recognizing specific activities (e.g. biking, walking, etc.), we target the novel task of recognizing user attention. To the best of our knowledge, our work is the first to consider this problem in a general way, not being limited to specific corner cases such as detecting when the phone is in a pocket or laying face-down on a table.

III. Proposed Method

We propose a system for detecting smartphone/tablet user attention automatically, using the data collected by common low-power sensors hosted in these devices. The most obvious solution would be to use the front-facing camera to periodically capture an image, then perform a face detection algorithm to infer the presence of users looking at the device screen. Conversely, we proposed an alternative method with several advantages. First and foremost, we only rely on data derived from extremely low-power sensors, rather than on the power hungry camera sub-system. Secondly, our approach is also preferable from a security perspective, as the camera does not need be constantly on in the background while the device is used, thus not creating new opportunities for malicious third parties to steal private user information.

Overall, the design flow at the basis of our method can be split into three main parts, as depicted in Figure 1. Phase 1 is the *generation of training data*, i.e. the collection and labelling of sensor data corresponding to scenarios in which users are/aren't paying attention to their device. Phase 2 consists of using these data to identify and train a classifier able to distinguish the two types of scenario based solely on sensor outputs. Finally, in Phase 3, the previously trained classifier is deployed on the device. In this work, we focus mainly on

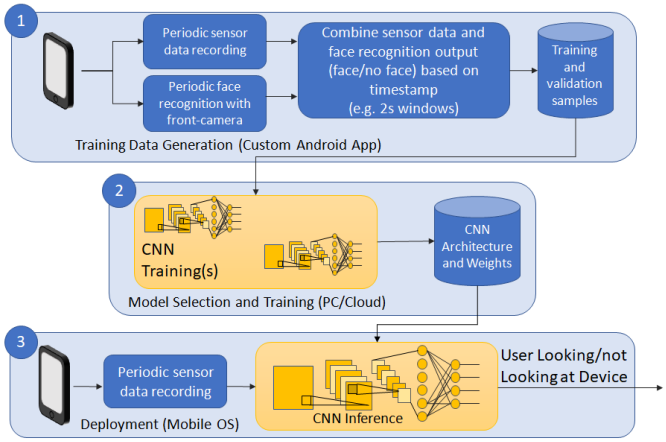

Fig. 1. Overall scheme of the proposed flow.

the first two phases, in order to prove the feasibility of our proposed approach. The performance of the final classifier is evaluated by simulations, and its actual deployment on the device will be the objective of our future work.

A. Training Data Generation

The goal of our method is to emulate the user attention detection that could be obtained by constantly monitoring the front-facing camera, without the energy and security issues of that approach. Therefore, the results of a camera-based system can be seen as a *golden reference* for those of our classifier. With this rationale, we can collect sensor readings associated with a camera-based user attention detection event and label them with the classification outcome (i.e. user looking/not looking at the screen) to transform the design of our camera-less classifier into a supervised-learning problem.

To perform this task, we designed an Android application that spawns several services to continuously collect data from sensors in background. At the same time, a separate service periodically activates the front-facing camera and takes a picture. The captured image is processed locally by Google's "FaceDetector" class, after being rotated multiple times to account for all possible device orientations. The timestamped sensor values and classification outcome are then anonymously uploaded to a cloud server to be used as training data.

We asked a pool of 20 participants to install and activate the application on their smartphones, both while not using their device and while performing common activities such as web browsing, messaging, fitness tracking, playing games, etc. After a few days, we collected a total of 12524 samples, each corresponding to one execution of the face detection algorithm. As expected, these samples were strongly unbalanced with respect to the two classes (device in use/not in use), with the latter accounting for 10473 samples, i.e. 83% of the total.

B. Model Selection and Training

The second phase of our flow corresponds to the identification and training of a classifier able to discern between samples corresponding to devices with and without user attention.

1) Feature Selection: In Phase 1, we collected data from all sensors available in each participant's device, including among others microphone, gyroscope, linear accelerometer, etc. However, before starting Phase 2, we performed a manual selection of "features" to be used as inputs for our classifier. This selection was driven by two main observations. Firstly, some sensors (e.g. gyroscope and linear accelerometer) are not available in all smartphones/tablets and using them would limit the applicability of our method to high-end devices. Secondly, the amount of data generated by the microphone for a given time window is orders of magnitude larger than that of the other sensors, hence using these data would inevitably increase the complexity of the classifier; moreover, using audio data poses privacy and security concerns similar to those occurring when using the camera, and definitely more critical than those of other sampled quantities (e.g. acceleration or ambient light). Based on these considerations, we finally trained our classifier using inputs from the following 4 sensors only:

- **Uncalibrated accelerometer:** data collected for the x, y and z axes, expressed in m/s^2.
- **Proximity**: data expressed as binary near/far events[1].
- **Ambient light:** data expressed in lux.
- **Multi-touch:** data expressed as touch/no touch events.

2) Data Pre-processing: The decision performed by our classifier is based on a time window of recent sensor readings. Given that all considered sensors generate readings with a frequency in the order of tens of Hz, we simplified the design of our classifier by pre-processing raw data. Specifically, we split the time-window into slices of 0.1s (i.e. 10Hz) and generated one datum per sensor for each slice. For periodically sampled sensors (accelerometer and light) each datum is simply the average of the readings in that slice. For event-based sensors (proximity and touch) the latest sampled value is maintained until the next event. For example, if a "near" event is captured by the proximity sensor in one slice, all following slices maintain that value until a "far" event is received. This eases the design of the classifier, as all sensors generate exactly the same number of inputs per time window, avoiding problems related to multi-modal classification, as explained in [3].

Notice that, for periodically sampled sensors, the Android OS only allows us to specify the sampling frequency as a "hint", so the data is not guaranteed to be exactly collected at that rate (e.g. in case of high OS load). For this reason, we actually sampled raw data at 20Hz and then averaged them as explained. Nonetheless, there were still samples that did not contain 1 value every 0.1s; those have been simply discarded.

3) Classifier Architecture: For selecting a machine learning model to use for our classification, we considered the conclusions drawn in [3] for a relatively similar task. As mentioned in Section II, the authors of that work showed that CNNs outperform other models for context detection. Consequently, we trained a very simple CNN to recognize user attention.

[1]The proximity sensors present in some devices return more than two values, corresponding to different distances. However, to maximize the portability of our method, all values different from 0 have been mapped to the "far" event.

Sensor data processed as explained in Section III-B2 are rearranged to form a virtual "image" of size 6x10N, where N is the length of the time window in seconds and each "pixel" is a sensor reading. Each row corresponds to one sensor, as shown on the left in Figure 2 and contains 10N readings. The order of rows is relevant, especially for the accelerometer. In fact, by placing the 3 axes data on adjacent rows, the local filters that compose Convolutional layers will be able to capture features that depend on the combination of acceleration values on different directions (e.g. device orientation).

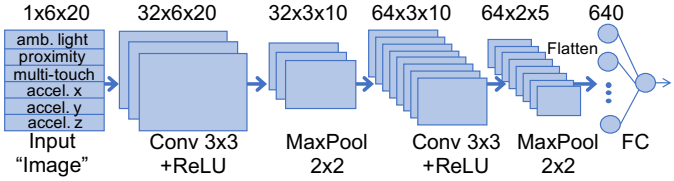

Fig. 2. Selected CNN Architecture.

The CNN shown in the rest of Figure 2 was selected after a design-space exploration phase (i.e. model selection), in which we evaluated and compared several variations of this architecture. Among others, we varied the number and type of layers, the number of feature maps/neurons in each layer, and the size of convolutional and max pooling features. Moreover, we also explored different training algorithm parameters such as the batch size, the number of training epochs, the learning rate and its variation policy, etc. The detailed results of this exploration are not reported for sake of space.

Different architectures have been compared using 10-fold Cross Validation, due to the relatively low number of samples, and using the F1-Score metric [8]. The final architecture was selected as the smallest (in terms of number of parameters) among those that yielded the highest F1-Score average on the 10-folds, i.e. larger models were discarded as they did not provide performance improvements.

Such architecture is composed of two Convolutional layers (Conv) with 3x3 kernels and Rectified Linear Unit (ReLU) activations, alternated with two 2x2 Max Pooling layers (MaxPool). The last MaxPool output is flattened and fed to a Fully Connected (FC) layer with 640 nodes ($64 \cdot 2 \cdot 5 = 640$) and with a Sigmoid activation function. The output of the FC layer is the estimated probability that the device is receiving user attention given the input. For all the details on the different CNN layers, the reader can refer to [8].

Notice that the selected architecture falls in the *feature concatenation* category according to the nomenclature used in [3]. This choice was made because inputs are sampled by all 4 selected sensors with similar rates, which are then rendered exactly equal by the preprocessing described in Section III-B2. *Modality specific* solutions are preferable when input quantities have different time dynamics, whereas they would only complicate the classifier architecture in this scenario.

The numbers on top of each layer in Figure 2 report its output size expressed as (# of feature maps)x(width)x(height). As evident by those numbers, a time window of N=2 seconds was

selected. This empirical choice represents a good compromise between classification complexity and performance. Indeed, although a larger window could allow the recognition of longer and more complex user actions, it would also significantly increase the input size, and hence the complexity of the CNN.

The CNN was trained using an Adam optimizer and the Binary Cross-Entropy loss function [8]. Importantly, we dealt with the class imbalance problem described in Section III-A by weighing each training example with the inverse of its class frequency. In other words, samples relative to devices *in use* and *not in use* were given different weights in the loss function, equal to $\frac{1}{0.17}$ and $\frac{1}{0.83}$ respectively. This corrects the inherent bias problem of class-imbalanced trainings, i.e. that the CNN would otherwise give more "importance" to errors on the *not in use* class due the larger number of samples belonging to it.

IV. Experimental Results

A. Classification Results

We trained and evaluated our CNN-based user attention classifier using the PyTorch framework in Python 3.6 [9]. The final CNN was trained with the parameters reported in Table I.

TABLE I
CNN training parameters.

# of Epochs	Mini-batch Size	Learning Rate	Weight Decay
200	100	3e-03	5e-07

The scores of the network have been computed as the average results of 10-fold Cross Validation (CV). The confusion matrix obtained with the trained CNN on the validation set is reported in Table II. The numbers in each matrix cell are fractional because they are the average over the CV iterations. A set of classification scores corresponding to such confusion matrix are reported in Table III. As shown by these numbers, the classification is more accurate when recognizing "Not Looking" samples (as indicated by the large specificity) than when recognizing "Looking" samples (slightly lower sensitivity). However, this is probably mostly due to the smaller number of training samples available for the positive class.

TABLE II
CNN confusion matrix.

Actual Value	Predicted Value		Total
	Looking	Not Looking	
Looking	161.8	42.4	204.2
Not Looking	45.4	1001.8	1047.2
Total	207.2	1044.2	

TABLE III
CNN scores. (Sensitivity: percentage of "Looking" samples correctly identified as such; Specificity: percentage of "Not Looking" samples correctly identified as such).

Accuracy [%]	Sensitivity [%]	Specificity [%]	F1-Score [%]
92.98	79.23	95.66	78.66

Fortunately, specificity is much more important than sensitivity for our energy management goals. In fact, smartphones/tablets waste energy when they are left in idle state, possibly with their display on, while no one is looking at them. A highly specific classifier will immediately identify this situation and transition the device to suspended state, thus significantly reducing the energy overheads. The opposite scenario, i.e. the automatic exit from suspended state when the phone starts to be actively used, would require a highly sensitive classifier. However, this functionality would mostly affect user experience rather than energy (in case of misclassifications the phone would not turn on) and is therefore less interesting for our purpose. In the following, we assume that the exit from the suspended state is managed normally by the user, for instance by pressing a button or tapping the screen.

Another possible malfunctioning of our system occurs when the device is actively used, but the classifier erroneously infers that it is not, and thus triggers a transition to suspended state. While this could occasionally worsen user experience, it would be very easy to avoid by simply creating OS "wake locks" for those scenarios in which the device should never be suspended (e.g. a video player could create a wake lock while a video is playing and remove it when it is paused).

Finally, a last possible concern about our classifier is that, rather than combining information from multiple sensors, the CNN is actually mainly driven by multi-touch events (an obvious indicators of user attention), and owes only to them its good performance. To verify this possibility, we trained the same CNN but *excluding* the multi-touch input. The final F1-Score obtained does decrease, but only marginally from 78.6% to 77%, proving that most useful information is actually provided by other sensors. A complete sensitivity analysis with respect to all four inputs is part of our future work.

B. Energy Models

In the rest of the section, we analyse the energy benefits and overheads of the proposed method. As mentioned in Section IV-A, we focus on a scenario in which a device is left in idle state and with the display on while unused.

In this scenario, an "oracle" policy would immediately perform the transition to suspended state. Calling P_{off} the power of the device in suspended state, and assuming that the period in which the device remains unused is significantly longer than the state transition time, such ideal policy would consume P_{off} for the entire idle period. We compare this ideal result and our camera-less detection against (i) a standard timeout-based power management policy, and (ii) a hypothetical method based on face detection. We assess the energy overheads of each approach through the following models.

1) Timeout-based power management: A straight-forward approach, implemented in most commercial devices is based on fixed (possibly user-defined) timeouts. There are normally two timeouts [10], one for turning off the display ($T_{to,1}$), and another to perform the transition to suspended state ($T_{to,2} > T_{to,1}$). Let us define P_{idle} as the power of a device in idle state, i.e. when the CPU and RAM are active, but no applications

are running and the display is off, and P_{disp} the additional power consumption of the display. The energy overhead of a timeout-based approach compared to the oracle policy is then:

$$E_{ovr,to} = T_{to,2} \cdot (P_{idle} - P_{off}) + T_{to,1} \cdot P_{disp} \quad (1)$$

2) Camera-based power management: One alternative is to use the camera subsystem to detect the presence of a face in front of the device and perform the transition to suspended state otherwise. In a real context, a face detection algorithm would have a non-zero probability of misclassification, which should be accounted for. However, since our method uses the camera-based classification as golden reference, we assume that the latter has a 100% accuracy. We assume that the presence of a face is sampled with a period T_s, and that the camera must be on for a time T_{cam} in order to perform auto-focus and capture a picture. During this time, the camera subsystem consumes P_{cam}. We also call P_{fd} the additional CPU power consumption (with respect to P_{idle}) required by the face detection algorithm and T_{fd} its average execution time. The energy overhead of this system is therefore:

$$E_{ovr,cam} = T_s \cdot (P_{idle} + P_{disp} - P_{off}) + T_{cam} \cdot P_{cam} + T_{fd} \cdot P_{fd} \quad (2)$$

3) CNN-based power management: As detailed in Section IV-A, our CNN-based classification has a probability $p = 0.9566$ of correctly inferring when a device is unused (the *Specificity* result of Table III). For a fair comparison, we assume that classifications are repeated with the same period T_s used for the camera-based solution. Moreover, we assume that different classifications do not have common inputs and can therefore be considered independent. This is true if T_s is greater or equal to the length T_{sens} of the data window required by the CNN. In this setting, the probability of correctly guessing the unused state in k tries can be modelled with a geometric distribution:

$$P(X = k) = (1 - p)^{(k-1)} \cdot p \quad (3)$$

Therefore, the average time before a correct classification is:

$$T_{avg} = T_s \cdot E(X) = \frac{T_s}{p} \quad (4)$$

Despite not using the camera, our approach requires that the sensors providing inputs to the CNN remain on for T_{sens}, consuming a power P_{sens}. Calling P_{cnn} and T_{cnn} the additional CPU consumption due to CNN inference and its execution time, the energy overhead of the CNN-based solution is:

$$E_{ovr,cnn} = T_{avg} \cdot (P_{idle} + P_{disp} - P_{off}) + T_{sens} \cdot P_{sens} + T_{cnn} \cdot P_{cnn} \quad (5)$$

C. Energy Analysis

Clearly, the exact power and time values for the quantities defined in Section IV-B depend on the considered device. In this section, we quantify the overheads of the different approaches using realistic assumptions, summarized in Table IV.

TABLE IV
REFERENCE TIME AND POWER VALUES FOR THE ENERGY ANALYSIS.

Time Values [s]						
$\mathbf{T_{to,1}}$	$\mathbf{T_{to,2}}$	$\mathbf{T_s}$	$\mathbf{T_{cam}}$	$\mathbf{T_{sens}}$	$\mathbf{T_{fd}}$	$\mathbf{T_{cnn}}$
10	30	2	1	2	≈0	≈0
Power Values [W]						
$\mathbf{P_{off}}$	$\mathbf{P_{idle}}$	$\mathbf{P_{disp}}$	$\mathbf{P_{cam}}$	$\mathbf{P_{sens}}$	$\mathbf{P_{fd}}$	$\mathbf{P_{cnn}}$
0.025	0.333	0.150	1.4	0	-	-

Initially, we consider the case in which both machine learning-based solutions perform a decision every $T_s = 2s$, so that $T_s = T_{sens}$, i.e. the length of the data window required by the CNN (see Section III-B). For the timeout-based policy, we set $T_{to,1} = 10s$ and $T_{to,2} = 30s$. Finally, we assume that the camera must stay on for $T_{cam} = 1s$ to take a picture.

For what concerns power data, we use figures reported in literature, referring to measurements performed on real devices. According to [11], the power consumption of a Google Nexus One in suspended state (i.e. the least consuming operational state, in which CPU and RAM are in low-power modes and all functionalities are disabled except for mobile connectivity) is $P_{off} \approx 25mW$. When the same device is in idle state, the consumption becomes $P_{idle} \approx 333mW$. An additional $P_{disp} \in [38 \dots 257]mW$ is consumed if the device display is on, depending on its brightness setting. For our analysis we assume an average level of brightness and consumption of $P_{disp} = 150mW$. In [12], the power consumption of the camera subsystem for a similar Nexus device during a low-resolution recording is measured to be $P_{cam} \approx 1400mW$. Results consistent with this figure are obtained in [13].

As explained in [14], accelerometer and proximity sensor are normally kept active during idle mode and are consequently embedded in P_{idle}. The same applies for the touch sensor, which must be active to manage events such as turning on/off the screen by double-tapping or similar actions. Lastly, the status of the ambient light sensor is typically linked to that of the display. However, the datasheet of a commercial sensor IC [15] reports an on-state power consumption of $0.65\mu A$, totally negligible compared to the previously reported values. Therefore, we assume $P_{sens} \approx 0$.

Similarly, we also neglect energy contribution of the two machine learning algorithms. The rationale in this case is that both algorithms are likely to complete in a time that is orders of magnitude smaller than the other operations involved in our analysis. Therefore, regardless of the CPU power consumption during classification, its *energy* impact would be negligible. To validate this assumption, we profiled a single classification by our CNN, executed in Python on a consumer laptop in single-thread CPU mode, which completed in $1.19ms$ on average. Even assuming a 10x slowdown on a mobile device, the energy for the classification would still be negligible compared to the other quantities involved in the models of Section IV-B.

Concerning the face-detection algorithm, its execution time will definitely be longer than the CNN inference, if only for the processing of a much larger input size (a high definition image). Thus, neglecting classification energy does not signif-

TABLE V
ENERGY ANALYSIS RESULTS.

Method	Reference Equation	Energy Ovr. [J]	Reduction w.r.t. Timeout [%]
Timeout	$E_{ovr,to} = 30 \cdot (0.333 - 0.025) + 10 \cdot 0.150$	10.74	-
Camera	$E_{ovr,cam} = 2 \cdot (0.333 + 0.150 - 0.025) + 1 \cdot 1.4 + 0$	2.316	78.4
CNN	$E_{ovr,cnn} = \frac{2}{0.9566} \cdot (0.333 + 0.150 - 0.025) + 0 + 0$	**0.958**	**91.1**

icantly alter the results of our analysis, and surely does not favour our approach.

The results of our energy analysis are reported in Table V. As shown, under the aforementioned assumptions, the camera-based approach reduces the energy overheads of 78.4% compared to a standard timeout. However, our CNN-based alternative achieves a significantly larger reduction (91.1%) with respect to timeout, and further reduces the overheads of 58.6% with respect to a system using the camera.

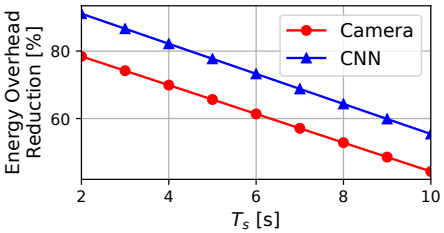

Fig. 3. Overhead reduction with respect to a timeout-based approach for different values of T_s.

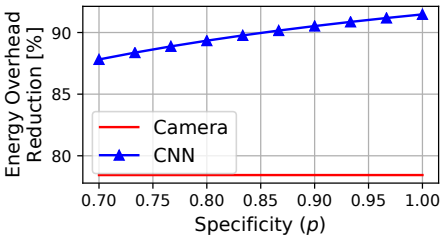

Fig. 4. Overhead reduction with respect to a timeout-based approach for different values of Specificity (p), with $T_s = 2s$.

Figures 3 and 4 show the variation of the energy gains with respect to the timeout policy, as a function of two design parameters: the classification period (T_s) and the specificity of the model (p). The first graph shows that even if the classification is performed more sporadically, both machine learning methods still provide significant benefits compared to a simple timeout; in particular, even with $T_s = 10s$, our classifier still reduces the energy waste by more than 50%. Moreover, the advantage over the camera-based method is not affected by the timeout value.

In Figure 4, the result of the camera-based approach is constant, as p is a parameter only of our method. This graph demonstrates that a user attention detector based only on low-power sensors would still provide significant benefits over a camera-based approach, even at relatively low accuracies, due to the elimination of the camera sub-system energy overheads. This result opens to further work aimed at simplifying the classifier, to further reduce the already limited CPU time spent for classification by the device.

V. CONCLUSIONS

We have proposed a novel way to perform smart power management in mobile devices. Our method is based on using a CNN classifier to determine whether users are paying attention to their device, and trigger power management actions accordingly (e.g. suspend the device and turn off its display).

We have shown that this method significantly reduces the energy waste when the device is unused but active, both compared to a standard timeout-based power management and to a solution that performs the same task using the front-facing camera. Future improvements of this work will include a more thorough design space exploration, to further simplify the classifier and/or improve its performance, and the actual deployment of the proposed system on a real mobile device.

REFERENCES

[1] F. Attal et al., "Physical Human Activity Recognition Using Wearable Sensors", Sensors 2015, 15, pp. 3131431338.

[2] N. Vallina-Rodriguez, J. Crowcroft, "The case for context-aware resources management in mobile operating systems", Springer Mobile Context Awareness, pp.97–113, 2012.

[3] V. Radu et al., "Multimodal Deep Learning for Activity and Context Recognition", Proc. of the ACM on IMWUT, Vol. 1, No. 4, Nov. 2017.

[4] L. Zhang et al., "Attention in convolutional LSTM for gesture recognition", Proc. NIPS'18.

[5] D. Jahier Pagliari et al., "Dynamic Bit-width Reconfiguration for Energy-Efficient Deep Learning Hardware", Proc. ISLPED 2018.

[6] D. Jahier Pagliari et al., "Energy-efficient Digital Processing via Approximate Computing", Smart Systems Integration and Simulation, pp.55–89, Springer, 2016.

[7] U. Christoph et al., "Context Detection on Mobile Devices", In Proc. CoSDEO, 2010.

[8] I. Goodfellow et al., "Deep Learning", MIT Press, 2016.

[9] A. Paskze et al., Automatic differentiation in PyTorch, In Proc. NIPS-W, 2017.

[10] https://source.android.com/devices/tech/power/platform_mgmt

[11] A. Carroll and G. Heiser, "An analysis of power consumption in a smartphone," In Proc. USENIXATC'10, pp.21–21.

[12] X. Chen et al., "How is energy consumed in smartphone display applications?," In Proc. HotMobile '13.

[13] S. S. K. Kasireddy and V. R. Bojja, "Measurements of Energy Consumption in Mobile Applications with respect to Quality of Experience", Master Thesis, Blekinge Institute of Technology.

[14] I. Koenig, A. Q. Memon and K. David, "Energy consumption of the sensors of Smartphones," Proc. ISWCS 2013, pp. 1-5.

[15] Maxim Integrated, "MAX44007, Low-Power Digital Ambient Light Sensor with Enhanced Sensitivity", Rev 1, 2011

[16] Z. Zhuang, K.-H. Kim, J. P. Singh, "Improving energy efficiency of location sensing on smartphones", in Proc. ACM MobiSys 10, pp. 315330, 2010.