Efficient Image Sensor Subsampling for DNN-Based Image Classification

ABSTRACT

Today's mobile devices are equipped with cameras capable of taking very high-resolution pictures. For computer vision tasks which require relatively low resolution, such as image classification, subsampling is desired to reduce the unnecessary power consumption of the image sensor. In this paper, we study the relationship between subsampling and the performance degradation of image classifiers that are based on deep neural networks (DNNs). We empirically show that subsampling with the same step size leads to very similar accuracy changes for different classifiers. In particular, we could achieve over $15\times$ energy savings just by subsampling while suffering almost no accuracy lost. For even better energy accuracy trade-offs, we propose AdaSkip, where the row sampling resolution is adaptively changed based on the image gradient. We implement AdaSkip on an FPGA and report its energy consumption.

1. INTRODUCTION

High-resolution cameras are prevalent on today's mobile devices. Apple's iPhone X is equipped with a 12-megapixel camera [1] and the latest Sony Xperia XZ has a dazzling 23-megapixel camera [2]. While the increasing consumer demand for high-quality photography justifies the need for a high-resolution camera, image sensors with over 10 million pixels are usually an overkill for many computer vision tasks. Take image classification as an example. For the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) [3], a popular 1000-class classification competition, competitors usually resize input images to 224×224. For simpler problems, such as the CIFAR-10 10-class classification problem [4], state-of-the-art classifiers achieve over 95% accuracy with input images of size 32×32. As the amount of computation required for a convolutional neural network (CNN) will roughly quadruple if the input image doubles its height and width, it is likely that the input sizes of classifiers will be kept as small as possible. That being said, a 5500×4200 image produced by a 23-megapixel appears disproportionately big compared to what we need today for common computer vision tasks.

The question then becomes how do we obtain images a with relatively low resolution from these over-sized image sensors. A straightforward way to obtain a is to take the full image and resample using bilinear kernels to create a smooth downsized image. However, energy-wise it is not the best solution. There exists an approximately linear relationship between the energy consumption of the image sensor and the number of pixels it samples [5]. As a result, it is desired to sample as few pixels as possible. But how much should we subsample? If we undersample, the aliasing effect causes distortions that might lead to degraded performances of the computer vision algorithms. If we oversample, we risk wasting energy by sampling unnecessary pixels. It is an important question to hardware and system designers, as it mandates the type of features and APIs that are exposed to application developers. It is also an important question to computer vision app developers, as its answer instructs them on how best to utilize the camera for the overall energy efficiency of mobile computer vision systems.

To bridge the gap between image sensor hardware and DNN-based image classification algorithms, we analyze the cause of errors when we subsample and propose clear and practical strategies to improve it. In particular, we show that subsampling, when done right, does bring about tremendous energy saving with negligible loss of accuracy. We do not intend to provide an exact answer to how much we should subsample. Instead, we show that by studying the performance degradation of just a few classifiers with subsampled input images, we can get a reasonable estimate of the performance degradation of image classification in general. To further reduce the energy consumption of the image sensor, we propose a minimal hardware modification to off-the-shelf image sensors. AdaSkip, the proposed enhancement, adaptively subsamples parts of the image based on the complexity. We show that the proposed method can achieve even better energy accuracy trade-offs than plain subsampling.

The contribution of this paper is neither proposing new image sensor architecture nor building more efficient computer vision algorithms. Rather, we take existing and widely-used designs from both sides and optimize from a system's perspective. To the best of our knowledge, we are the first to a) systematically study the relationship between subsampling and performance degradation of DNN-based image classification; b) propose a low-footprint hardware extension to off-the-shelf CMOS image sensors for more efficient subsampling in the context of efficient image classification.

We organize the rest of the paper as follows. Section 2 introduces prior art from the images sensor and deep DNN optimization communities. Section 3 briefly describes the operation of CMOS image sensors and our energy model. Section 4 analyzes the negative effect subsampling has on DNNs and motivates the problem. Section 5 details both of our proposed methods. Section 7 evaluates the effectiveness of our methods.

Submitted to ISLPED 2018

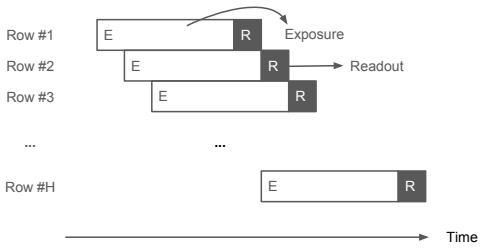


Figure 1: CMOS image sensor operation with rolling shutter.

2. RELATED WORK

CMOS Image Sensors. It is well known that the analog readout chain consumes a significant amount of power. Thus there has been a continuous effort in creating *specialized* image sensors that take reduced amount of sampling. Notable approaches include the use of DCT proposed by Kawahito *et al.* [6], predictive coding proposed by Leon-Salas *et al.* [7], Quadtree Decomposition algorithm proposed by Artyomov *et al.* [8], and compressive sensing proposed by Oike *et al.*[9]. Kemeny *et al.* also proposed an image sensor that allows programmable readout of pixels at arbitrary locations and various resolutions [10].

LiKamwa *et al.* studied the energy proportionality of off-the-shelf image sensors and proposed optimization strategies for simple computer vision tasks [5].

In all of the cases above, the primary constraint is image quality. In our work, however, we look at the effect of subsampling on the particular task of image classification. For the hardware portion of our contribution, instead of trying to derive an ad-hoc image sensor architecture, we base our techniques solely on subsampling by skipping pixels, a feature that is widely available on today's mobile image sensors.

Neural Network Design. DNNs are extremely power hungry. The quadratic relationship between computation and input size plays a critical role in optimizing the power consumption of DNNs. Huang *et al.* proposes the multi-scale DenseNet architecture, where the model dynamically adopts early size-reducing pooling layers for easier inputs [11].

Other researchers have also proposed to reduce the computation of neural networks by using different image sensor designs. Chen *et al.* have proposed to approximate the first layer of CNNs using image sensors with angle sensitive pixels [12]. LiKamwa *et al.* proposed to move early layers in CNNs to the analog domain to save energy.

In comparison to the designs mentioned above, our work targets a broader audience. We base our experiments on generic neural network models. And the main target of our study is how the most widely used image sensor architecture could best be utilized in order to serve need of neural networks.

3. CMOS IMAGE SENSORS

In this section, we describe the architecture and operation of the most commonly used mobile CMOS image sensors. We then describe the model based on previous energy characterizations of off-the-shelf units [5].

3.1 Architecture and Operation

Today's mobile CMOS image sensors usually consist of pixel arrays, an analog signal chain and various digital processing and I/O logic. The analog chain contains ADCs, amplifiers, and bias adjusting circuits and it is known to be the major source of power consumption. A common design choice for image sensors is to use *column-parallel readout*, where one pixel column share one signal chain. In this architecture, the pixels are usually read out row by row.

Rolling electronic shutters are widely used to control the exposure. During the operation of a rolling shutter, only one row or column of pixels read out at a time. The operation fits nicely with the column-parallel readout architecture. When one row finished exposure, the readout logic starts the processing of that row. At the same time, the subsequent row is in the process of exposure, waiting for its turn for pixel readout. The whole process is depicted in Figure 1.

There are usually two supported modes of subsampling: *skipping* and *binning*. In skipping mode, the sensor skips the entire pixel readout chain of pixels at certain locations. In binning mode, pixel values are averaged before output. We mainly consider the skipping mode, since it is the mode that enables the most energy savings.

3.2 Energy Model

We mainly use LiKamWa *et al.*'s energy characterization on commercial image sensors[5]. They proposed to estimate the energy consumption of image sensors using the following equation:

$$E = P_{idle}t_{idle} + P_{active}t_{active}$$

P_{idle} and t_{idle} represents the power and time when no pixel readout is occurring, i.e. when the first row is going through exposure. Once the pixel readout starts, the system switches to the active mode with higher power consumption. The length of the readout t_{active} is linear to the number of pixels sampled.

4. PROBLEMS WITH SUBSAMPLING

Given a neural network that requires inputs of size $H \times W$, it is most desirable if we directly subsample the pixel arrays to create an $H \times W$ image. However, images naturally contain a variety of high-frequency components its original form. Subsampling images can easily give rise to strong *aliasing* effect.

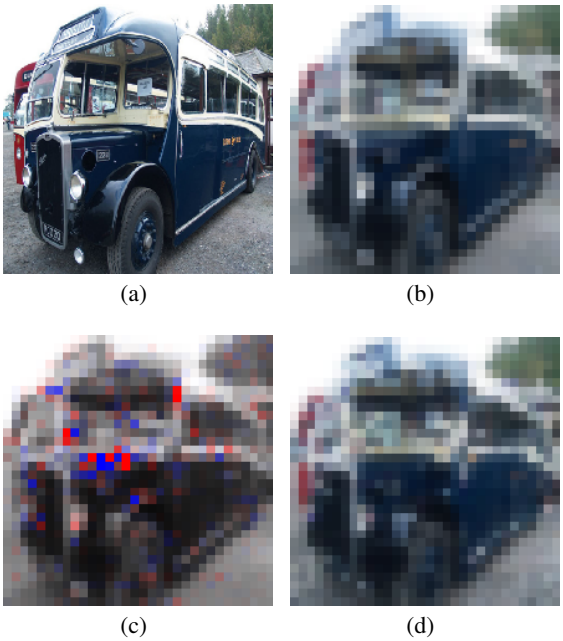
Neural networks have very unpredictable behaviors when they encounter input data derived from distributions that are different from what they are trained on. Goodfellow *et al.* demonstrated that extremely small modifications in the directions to increase the loss can effectively fool neural networks [13]. From the neural networks' perspective, the process of subsampling creates input data of a slightly different distribution. It is thus likely that neural networks do not behave correctly in these situations.

Table 1: Effects of direct subsampling on 10-class classifiers with different input sizes

Subsampling method	Accuracy loss (%)
32×32, Direct subsampling	-48.6 $\pm$ 6.4
32×32, Smoothed (radius=0.3)	-36.6 $\pm$ 5.9
32×32, Smoothed (radius=0.65)	-9.3 $\pm$ 1.3
32×32, Smoothed (radius=0.9)	-50.2 $\pm$ 5.8
64×64, Direct subsampling	-31.7 $\pm$ 8.3
64×64, Smoothed (radius=0.55)	-6.1 $\pm$ 1.3

To verify our conjecture, we tested direct subsampling on 10-class image classifiers with two different input sizes. Section 6 explains the experimental setup in more details. To create an image of the target input size, we directly subsample (by skipping pixels) from a 512×512 image. Then we feed it to 10 different classifiers, with classification accuracies averaging 86.9% and 90.6%

for 32×32 and 64×64 classifiers respectively. Fearing that the extreme variation in neighboring pixels might be adversely affecting the performance of the classifiers, we further added a low pass Gaussian filter to smooth out the images. We used binary search to find a radius that performs relatively well. Table 1 shows the mean accuracies and corresponding 95% confidence intervals. We observe substantial accuracy drop when direct subsampling is applied. Smoothing helps, but in the best case, we still see an average of 9.3% accuracy drop for 32×32 classifiers and 6.1% accuracy drop for 64×64 classifiers.



(a) (b) (c) (d)

Figure 2: Errors caused by subsampling. (a) The original image of a bus; (b) 32×32 resampled image using bilinear kernel; (c) visualized source of error I in channel "R". Red dots are pixels that increase loss, and blue dots are those that decrease loss. The less transparent the color, the larger impact a pixel has; (d) 32×32 smoothed subsampled image.

To further study reasons behind the misclassifications, we study the effect each pixel has on the final loss function. To quantify the effect, we define the following measurement given loss function L, bilinearly resampled image $\mathbf{X}$ and smoothed subsampled image $\mathbf{X}'$:

$$\mathbf{I} = \frac{\partial L}{\partial \mathbf{X}'} \odot (\mathbf{X}' - \mathbf{X})$$

$\frac{\partial L}{\partial \mathbf{X}'}$ represents how sensitive the loss function to the change of individual pixel. It could be positive or negative, where increasing pixel value causes loss increase or decrease, respectively. $\odot$ is the Hadamard (element-wise) product. $(\mathbf{X}' - \mathbf{X})$ represents the change from the standard image to the subsampled. Intuitively, large changes in the sensitive pixels will inflate the loss and cause the classifier to misclassify.

We studied many misclassified samples and selected an example image of a bus as shown in Figure 2. Although area inside the windowshield in the subsampled image appears somewhat jagged, it is not contributing much in terms of error. It is the edge the windshield, where the color abruptly changes, that is contributing the most to the classification error. Apparently failing to sample the right color during subsampling distorted the appearance of the window frame and caused the classifier to misclassify. Above is a typical example where smoothing cannot recover the information lost during subsampling. The *only* way to mitigate the problem is to sample at a higher resolution to include more information.

REMARK 1. *Subsampling may distort key features that image classifiers use, causing unrecoverable classification performance degradation.*

As a side note, if there exists a classifier that is trained on directly subsampled images, then the problem of subsampling becomes trivial. However, to our knowledge, all of today's classifiers are trained on smooth images that are interpolated at least linearly when downsizing from a full resolution image. In addition, we carried out some classification experiments with bilinearly downsized and directly subsampled images. It appears that because of the substantial aliasing effect present among directly subsampled images, classifiers trained on those appear to overfit easily and do not perform as well. Further, from a system's perspective, we should not make any assumptions about the image classifiers. We thus restrict our discussion in the general case where the classifier is trained on images bilinearly resampled from full-resolution ones, while we tried to supply the classifier with subsampled image and maintain its performance.

5. PROPOSED METHOD

To avoid confusion, we are going to use the term "two-step subsampling" to describe the process of subsampling to create input images of specific sizes for a classifier. We formally define this process in the next subsection. Following the definition, we described our proposed adaptive sampling strategy, AdaSkip. For AdaSkip requires minimal hardware support. We discuss an alternative hardware design in the last subsection.

5.1 Two-Step Subsampling

Let $H_o \times W_o$ be the resolution of the image sensor. The image sensor also supports a list of K subsampling steps, $\mathbb{S} = \{s_1, s_2, \ldots, s_K\}$, where s_i represents the number of pixels among which one is sampled. We use `subsample(`$\mathbf{X}, s$`)` to represent the process of sampling with step s, both horizontally and vertically. $\mathbf{X}$ is of size $H_o \times W_o$, so the resulting image is of size $\frac{H_o}{s} \times \frac{W_o}{s}$. Instead of of subsampling the whole image $\mathbf{X}$, we use the same expression to represent the subsampling of one row $\mathbf{x}$, `subsample(`$\mathbf{x}, s$`)`.

To subsample to create images of specific size $H_t \times H_w$, we first command the image sensor to subsample with step s, where $\frac{H_o}{s} > H_t, \frac{W_o}{s} > W_t$. Then, taking the output from the sensor, we apply resampling (in software) using kernel function f `resample(`$\mathbf{x}, f$`)` to create image of the target size. Typical kernel function include bilinear, Bell, Bicubic, etc. We use bilinear kernel in our experiments. The procedure is described in Algorithm 1.

Algorithm 1 Two-Step Subsampling

Input: $\mathbf{X}$, the original image, of size $H_o \times W_o$; s, subsampling step size; f, the kernel used in software resampling.
Output: $\mathbf{X}'$, the image of desired size $H_t \times W_t$

1: $\mathbf{X}' \leftarrow$ `subsample(`$\mathbf{X}, s$`)` ▷ Hardware subsampling
2: $\mathbf{X}' \leftarrow$ `resample(`$\mathbf{X}', f$`)` ▷ Software resampling
3: **return** $\mathbf{X}'$

5.2 AdaSkip

Since undersampling distorts essential image features, we should sample at a rate that is high enough to keep the features in tact. However, not all parts of the image need a high sampling rate. It's more desirable to adaptively change the rate of sampling. The process of progressive exposure and readout in rolling shutters creates slacks after each row. We could use the slacks for making decisions on the step size used for the subsequent rows, based on the information that we obtain from the current row. Using this information, we could speculate on whether to sample them using high resolution or low resolution. With the varying granularity within one frame, the resulting image will look very undesirable aesthetically. However, deep learning algorithm might be able to look at the resulting image in a different light. Since the process of deciding what resolution to sample is adaptive, we coined the name AdaSkip, which stands for adaptively skipping rows.

In computer vision, the *gradient* of an image is a very fundamental element in feature engineering. It is usually defined in two terms, the gradient along the x direction G_x and the gradient along the y direction G_y.

$$G_x(\mathbf{X}) = |\frac{\partial \mathbf{X}}{\partial x}|$$

$$G_y(\mathbf{X}) = |\frac{\partial \mathbf{X}}{\partial y}|$$

Image gradients have an extensive range of usages. For one, it is commonly used as building blocks for edge features. As a direct feature, it is also used to determine the complexity of different parts of the images in seam carving. As a simple indicator of the complexity of the images, it is adopted in our algorithm to decide the sampling resolution. However, since we only have the information about the current row sampled, we will only use the gradient along the x-direction G_x. Algorithm 2 details our approach. We use $G_x(\mathbf{x_i})$ to represent the gradient in the row $\mathbf{x_i}$.

Algorithm 2 The AdaSkip algorithm

Input: $\mathbf{X}$, the original image, of size $H_o \times W_o$; s_h, s_l, the number of pixels to skip, $s_h < s_l$; thd, the threshold to determine which subsampling mode to use;
Output: $\mathbf{X'}$, the image of desired size $H_t \times W_t$

1: $step \leftarrow s_h$
2: Initialize $counter$
3: **for** each row $\mathbf{x}_i'$ **do** ▷ The main algorithm in hardware
4: **if** $counter = step$ **then** ▷ Sample and decide resolution
5: $\mathbf{x_i}' \leftarrow \texttt{subsample}(\mathbf{x_i}, step)$
6: $counter \leftarrow 1$
7: **if** $\max(G_x(\mathbf{x_i}')) > thd$ **then**
8: $step \leftarrow s_h$
9: **else**
10: $step \leftarrow s_l$
11: **end if**
12: **else** ▷ Skip the row
13: $counter \leftarrow counter + 1$
14: **end if**
15: **end for**
16: $\mathbf{X}' \leftarrow$ collections of rows $\mathbf{x_i}$
17: **return** $\mathbf{X}'$

The high-level idea of the algorithm is extremely straightforward: use a lower resolution if the current row is not complex, else use a higher resolution. There are a few details that are not reflected in the algorithm. First, in order to make sure that the thd is the same for two resolutions (two rows of different sizes), the denser row should be subsampled in the calculation of $G_x(\mathbf{x_i})$ to match the sparser one. Second, since the subsampled rows are of different sizes, eventually they should be unified (as stated in line 16) before further resampled. That could be done by manipulating the column memories in hardware or using the software.

5.3 AdaSkip Hardware Extension

Since the AdaSkip algorithm varies sampling rate per row, it can only be implemented as part of the images sensor hardware. The core logic is relatively simple, and it mainly involves line 7 in Algorithm 2. The calculation of gradient involves adders and converting two's complement representations. Comparators are next for finding the maximum value. Both components are already present in current image sensor design. Adders exist in digital gain adjustment circuits, and comparators in SAR based ADCs. To reduce the reduce the energy of the design, we could simply lower the precision of the arithmetic computation.

6. EXPERIMENTAL SETUP

6.1 Dataset

ImageNet is one of the most widely used datasets in image classification [14]. The subsets that is used for ImageNet Large Scale Visual Recognition Challenge (ILSVRC) contains 1,461,406 images from 1000 classes [3]. It contains object classes of very different nature and granularity, including classes ranging from "warplane" to a specific dog breed called "Irish water spaniel". All the images are available in its original JPEG format. The median number of pixels in the images is around 200,000. Many other datasets exist in the field of image classification, but they rarely contain images of the original sizes. Given the popularity, scale and data format, the ImageNet dataset is the best choice for our experiments.

6.2 Camera Configuration

Due to resolution limitations of the dataset, it is not possible to conduct experiments to reflect the performance on today's high resolution cameras. We decide to simulate the behavior of an image sensor with resolution 512×512. For subsampling, we assume that when the supported skip steps are s =1, 2, 4 and 8. We assume images are taken in the outdoor condition, with relatively less exposure time and thus less t_{idle}. In this case, the majority power consumption occurs during the active mode.

6.3 Image Classifiers

Table 2: Classifiers used in our experiments

Input size	# Classes	Quantity	Accuracy (%)
32×32	10	10	86.9 ± 3.3
32×32	20	10	81.1 ± 2.0
32×32	30	5	78.9 ± 2.3
48×48	10	10	87.3 ± 4.2
48×48	20	5	86.0 ± 3.3
48×48	30	5	84.9 ± 2.7
64×64	10	10	90.6 ± 2.1
64×64	20	5	88.0 ± 3.5

To reenact a reasonable ratio between the image sensor resolution and classifier input size, we experiment on classifiers with 32×32, 48×48, and 64×64 input images. We randomly sample 10, 20 or 30 classes from the 1000 classes, and train 5 or 10 classifiers to form the population of classifiers with the corresponding

configuration [1]. All of the classifiers use the ResNet34 architecture [15], trained with over 150 epochs. We use SGD with a momentum of 0.9. The learning rate starts at 0.1 and is reduced to 1/10 the value every 50 epochs. Table 2 lists the classifiers that we used. In our experiments, we observed that some of the test images produce low confidence scores. In those cases, a small change in the input may trigger an arbitrary change in the prediction. We filter out those images with original confidence score below 0.8 when we compare different subsampling methods.

The randomly sampled classes have varying levels semantic similarities. Deng *et al.* showed, quite intuitively, that classes of closer semantic meanings are harder to distinguish [16]. Our experiments confirmed the observation. Take the 32×32, 10-class classifiers as an example. The worst performing classifier with only 80.2% accuracy tries to classify, among other classes, three types of amphibians: `tailed frog`, `common iguana` and `whiptail`. The best performing classifier achieves 92.2% accuracy on classes that are relatively disparate in meanings. Using these classifiers of very different nature as well as using different input sizes are all parts of our efforts to make our conclusions more generalizable.

7. EVALUATION

We start the evaluation by presenting the performance characteristics of all our classifiers when two-step subsampling is used. Next, we demonstrate the AdaSkip algorithm by first visualizing it, and then show its performance of it with respect to energy and accuracy.

7.1 Two-Step Subsampling

Table 3: Accuracy loss caused by two-step subsampling with different step sizes on different classifiers

Energy savings (step) / Classifier	56.0× ($s = 8$)	15.5× ($s = 4$)	3.97× ($s = 2$)
32×32, 10-class	-1.6 ± 0.6	-0.1 ± 0.2	0
32×32, 20-class	-1.3 ± 0.4	-0.1 ± 0.1	0
32×32, 30-class	-2.7 ± 0.4	-0.0 ± 0.2	0
48×48, 10-class	-3.0 ± 0.9	-0.4 ± 0.3	0.0 ± 0.1
48×48, 20-class	-4.4 ± 1.8	-0.2 ± 0.1	0
48×48, 30-class	-4.1 ± 0.7	-0.4 ± 0.3	0
64×64, 10-class	-30.5 ± 9.0	-0.2 ± 0.2	0
64×64, 20-class	-36.5 ± 9.5	-0.4 ± 0.5	0

Table 3 shows the accuracy loss using different steps s for subsampling. Different columns represent different step sizes and on the top row we also showed the relative energy savings. The gray areas represent cases where after subsampling, the image is less than twice (in both width and height) the size of the required input. Comparing to Table 1, even just doubling the sampling rate substantially reduces the accuracy loss. Further, we observe that the accuracy loss is correlated much more with the step size than with what classifier it is. It appears that as long it is at least $2\times$ larger than the target resolution of the classifiers, the same resolution after subsampling causes similar accuracy loss to all classifiers. These findings echo with the analysis from Section 4, where we argue that information loss at the subsampling step plays a crucial role in determining the performance degradation of image classifiers.

[1]Due to the time limit, we were not able to 30-class 64×64 classifiers

This characteristic has an interesting implication for system designers. Given a subsampling step size s, one could estimate the accuracy loss of potentially all classifiers using just a few of them. Based on this characteristics, hardware providers, for example, could present guidelines to app developers on the best subsampling strategy to use for different target image sizes needed by classifiers.

REMARK 2. *Regardless of what classifier it is, the performance degradation of a classifier given subsampled input images is largely decided by the subsampling step size.*

7.2 AdaSkip Subsampling

7.2.1 Visualization

(a) (b) (c) (d)

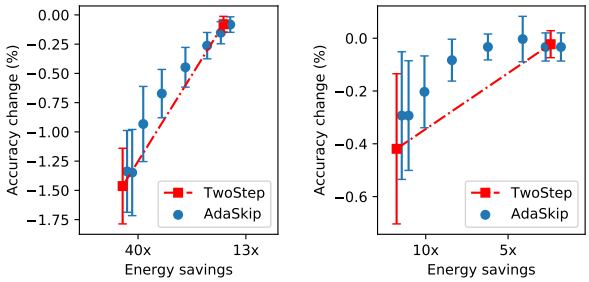
Figure 3: Visualizing AdaSkip-based subsampling. (a) The original 512×512 image; (b) the output of the AdaSkip algorithm, where the greyscale parts are rows sampled with $s_l = 8$ and the colored part are rows sampled with $s_h = 4$; (c) 32×32 output images from two-step subsampling with $s = 4$ and bilinear resampling; (d) 32×32 images obtained by bilinear resampling the output in (b).

Figure 3 visualizes the simulated subsampling procedure of the AdaSkip algorithm. The image belongs to the class `black stork`. For AdaSkip, we set the threshold for switching between the higher resolution $s_h = 4$ (equivalent to resolution 128×128), the lower resolution $s_l = 8$ (equivalent to resolution 64×64), and threshold to 96. To visualize the effect of using different subsampling resolutions, we resort to using grayscale and colored parts. As a result, as shown in Figure 3(b), a few rows on the top depicting grass and roughly the entire lower half are sampled using the lower resolution. Figure 3(c) depicts the output image from a two-step subsampling procedure. The first step is the hardware subsampling with $s = 4$ (resulting in a 128×128 image), and the second step is software bilinear resampling. Figure 3(c) shows the results from bilinear resampling the image in 3(b). Comparing 3(d) with 3(c), we barely see any difference. The resulting image is indeed correctly classified by the classifier.

7.2.2 *Hardware Energy Consumption*

We assume our hardware support is an auxiliary to an existing image sensor. Therefore our estimated power consumption is the upper bound of the additional power required to perform AdaSkip. Our implementation on a Spartan-6 LX45 field-progrmmable-gate-array (FPGA) shows the power overhead of the 128-pixel ($s_l = 4$) AdaSkip hardware support is approximately 13.5 mW. We claim that this overhead can be significantly reduced if AdaSkip hardware support is implemented on ASIC or co-optimized with the image sensor. Kuon *et al.* estimate the power gap between ASIC and FPGA implementations of logic only designs could be as large as $5.7\times - 52\times$ [17]. We estimate the power consumption of AdaSkip hardware support to be 2.4 mW or lower on ASIC, indicating less than 1% additional power in our energy model comparing to an image sensor without AdaSkip. In an actual image sensor, we argue that there is no need for more than 600-pixel AdaSkip support, as rarely will a classification problem needs that large of a resolution. Without considering the possibility of using memory and serializing, that amounts to less than 5% of additional power. We use that as the overhead in the following discussion of energy.

7.2.3 *Energy Accuracy Trade-Off*



(a) (b)

Figure 4: Energy accuracy trade-offs using AdaSkip. (a) 32×32 **classifiers with** $s_l = 8, s_h = 4$**; (b)** 48×48 **classifiers with** $s_l = 4, s_h = 2$

Simply by two-step subsampling, we already saved significant amount of energy. To take a step further, we apply AdaSkip in this setting. Figure 4 shows the accuracy changes using different thresholds in AdaSkip. In the figure, the y-axis is the accuracy change, and the x-axis is the relative energy consumption. Blue dots represent the results from AdaSkip, where each point represents a threshold (multiples of 32). Red square represents the baselines using two-step subsampling for s_l (left) and s_h(right) respectively.

When the threshold is 0, AdaSkip degrades to two-step but with additional hardware overhead, thus falls below the baseline. However, for almost all the other points, AdaSkip constantly achieve better energy accuracy trade-offs. In the case of Figure 4(b), sampling at $s = 2$ sustained an average of 0 accuracy lost. That is equivalent to $4\times$ energy savings. Using AdaSkip, we can boost the energy savings to around $7\times$ without losing any accuracy.

8. CONCLUSION

We present a study on how subsampling affects the performance of DNN-based image classifiers. We first demonstrate that one could achieve over $15\times$ energy savings just by subsampling while suffering almost no accuracy lost. Then we empirically show that the accuracy lost can be predicted as an approximate function of subsampling step size, regardless of what classifier it is. To achieve better accuracy when subsampling aggressively, we propose the AdaSkip algorithm. We implement AdaSkip on an FPGA and estimate that the overhead in a real image sensor to be less than 5%.

9. REFERENCES

[1] "iPhone X - technical specifications." https://www.apple.com/iphone-x/specs/. Accessed: 2018-03-01.

[2] "Xperia™XZ specifications - Sony mobile." https://www.sonymobile.com/us/products/phones/xperia-xz/specifications/. Accessed: 2018-03-01.

[3] O. Russakovsky and et al., "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, 2015.

[4] A. Krizhevsky and G. Hinton, "Learning multiple layers of features from tiny images," Master's thesis, Department of Computer Science, University of Toronto, 2009.

[5] R. LiKamWa, B. Priyantha, M. Philipose, L. Zhong, and P. Bahl, "Energy characterization and optimization of image sensing toward continuous mobile vision," in *MobiSys*, pp. 69–82, 2013.

[6] S. Kawahito, M. Yoshida, M. Sasaki, K. Umehara, D. Miyazaki, Y. Tadokoro, K. Murata, S. Doushou, and A. Matsuzawa, "A cmos image sensor with analog two-dimensional dct-based compression circuits for one-chip cameras," *J. Solid-State Circuits*, vol. 32, no. 12, pp. 2030–2041, 1997.

[7] W. D. Leon-Salas, S. Balkir, K. Sayood, M. W. Hoffman, and N. Schemm, "A CMOS imager with focal plane compression," in *ISCAS*, 2006.

[8] E. Artyomov and O. Yadid-Pecht, "Adaptive multiple-resolution CMOS active pixel sensor," *IEEE Trans. Circuits Syst.*, vol. 53-I, no. 10, pp. 2178–2186, 2006.

[9] Y. Oike and A. E. Gamal, "CMOS image sensor with per-column $\Sigma\Delta$ ADC and programmable compressed sensing," *J. Solid-State Circuits*, vol. 48, no. 1, pp. 318–328, 2013.

[10] S. E. Kemeny, R. Panicacci, B. Pain, L. H. Matthies, and E. R. Fossum, "Multiresolution image sensor," *IEEE Trans. Circuits Syst. Video Techn.*, vol. 7, no. 4, pp. 575–583, 1997.

[11] G. Huang, D. Chen, T. Li, F. Wu, L. van der Maaten, and K. Q. Weinberger, "Multi-scale dense convolutional networks for efficient prediction," in *ICLR*, 2018.

[12] H. G. Chen, S. Jayasuriya, J. Yang, J. Stephen, S. Sivaramakrishnan, A. Veeraraghavan, and A. C. Molnar, "ASP vision: Optically computing the first layer of convolutional neural networks using angle sensitive pixels," in *CVPR*, pp. 903–912, 2016.

[13] I. J. Goodfellow, J. Shlens, and C. Szegedy, "Explaining and harnessing adversarial examples," in *ICLR*, 2015.

[14] J. Deng, W. Dong, R. Socher, L. Li, K. Li, and F. Li, "Imagenet: A large-scale hierarchical image database," in *CVPR*, pp. 248–255, 2009.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *CVPR*, pp. 770–778, 2016.

[16] J. Deng, A. C. Berg, K. Li, and F. Li, "What does classifying more than 10, 000 image categories tell us?," in *ECCV*, pp. 71–84, 2010.

[17] I. Kuon and J. Rose, "Measuring the gap between fpgas and asics," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 26, no. 2, pp. 203–215, 2007.