A HYBRID APPROACH TO OFFLOADING MOBILE IMAGE CLASSIFICATION

J. Hauswald[⋆], *T. Manville*[⋆], *Q. Zheng*[⋆], *R. Dreslinski*[⋆], *C. Chakrabarti*[†] *and T. Mudge*[⋆]

[⋆]EECS Dept, University of Michigan, Ann Arbor
[†]School of ECEE, Arizona State University, Tempe

ABSTRACT

Current mobile devices are unable to execute complex vision applications in a timely and power efficient manner without offloading some of the computation. This paper examines the tradeoffs that arise from executing some of the workload onboard and some remotely. Feature extraction and matching play an essential role in image classification and have the potential to be executed locally. Along with advances in mobile hardware, understanding the computation requirements of these applications is essential to realize their full potential in mobile environments. We analyze the ability of a mobile platform to execute feature extraction and matching, and prediction workloads under various scenarios. The best configuration for optimal runtime (11% faster) executes feature extraction with a GPU onboard and offloads the rest of the pipeline. Alternatively, compressing and sending the image over the network achieves lowest data transferred (2.5× better) and lowest energy usage (3.7× better) than the next best option.

Index Terms— mobile computing, offloading, image classification, energy management

1. INTRODUCTION

Image classification in computer vision applications can be divided into feature extraction and machine learning based prediction. In computer vision, machine learning algorithms attempt to describe the content of images based on the features in the image. This typically requires training on a large set of images to make predictions about new images. Machine learning is used to predict if a specific object (or class of objects) is present in an image. The accuracy of the prediction is determined by the machine learning algorithm, the amount of training data available, and the quality of the features. For mobile platforms, machine learning algorithms require prohibitive amounts of computation and storage. Training on a mobile device is infeasible, in today's technology, and is usually offloaded to a cloud server. Google Glass for example, offloads classification and recognition to Google's data centers [1].

Figure 1 shows the conceptual stages of the image classification pipeline. As technology improves, more of the pipeline will be able to run locally, allowing applications to run without relying on a cloud connection, should a network connection be unavailable. Feature extraction and machine learning predictions must meet mobile device computation and energy budgets. Moving feature extraction onboard in a hybrid configuration will eliminate the need to transmit an entire image, which reduces the data required to offload. Currently, mobile developers in computer vision are working to find ways to reduce runtime, lower network usage and limit energy consumption. For example, the EFFEX [2] processor was developed to improve the performance of feature extraction algorithms on energy-constrained mobile embedded platforms. Several attempts have also been made to reduce network usage employing energy-efficient lossy image compression before transmission [3, 4]. Huang et al. compare the performance and power characteristics of 4G LTE, WIFI and 3G during transmission [5]. Gordon et al. developed COMET, a framework to offload computation for multithreaded kernels that achieved a speedup of 2.9× and energy savings of 1.51× on WIFI [6]. Maui [7] studies mobile offloading by making decisions at runtime where it should execute the code stages. Finally, CloneCloud [8] decides where to offload computation by profiling components of the kernels before runtime. Our work analyzes the computations needed throughout the pipeline, allowing us to identify which stages can be offloaded. Our work contributes the following:

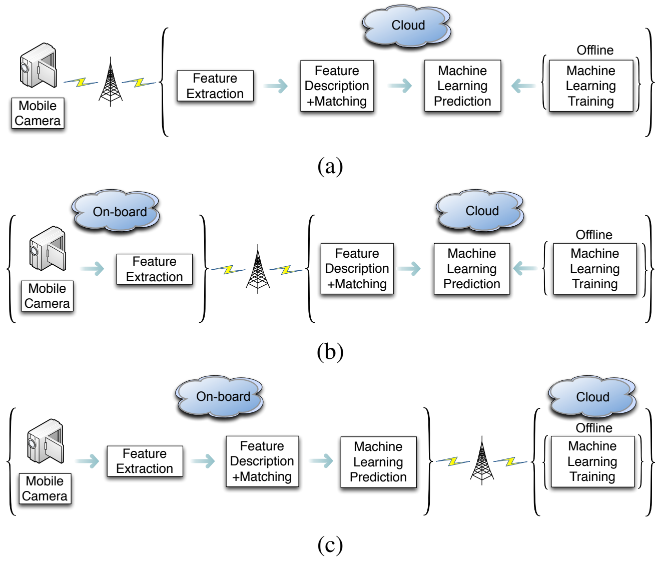

Cloud
Offline
Mobile Camera
Feature Extraction
Feature Description +Matching
Machine Learning Prediction
Machine Learning Training
(a)
On-board
Cloud
Offline
Mobile Camera
Feature Extraction
Feature Description +Matching
Machine Learning Prediction
Machine Learning Training
(b)
On-board
Cloud
Offline
Mobile Camera
Feature Extraction
Feature Description +Matching
Machine Learning Prediction
Machine Learning Training
(c)

Fig. 1: The computer-vision classification pipeline: (a) current (b) near-future (hybrid) and (c) far-future mobile pipelines.

- Characterize the data transfers between stages in the computer vision prediction pipeline
- Analyze the effects of image preprocessing (compression and resizing) on runtime and prediction accuracy
- Identify where to run the stages of the pipeline to improve runtime, lower network usage and limit energy consumption

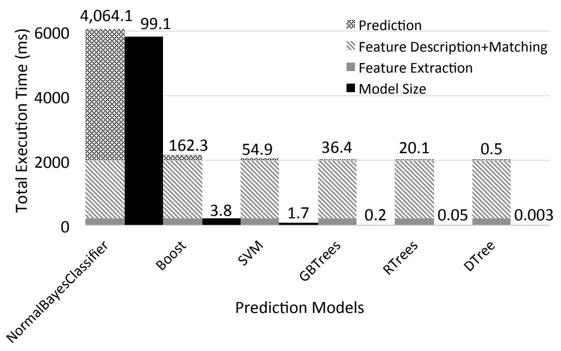


Fig. 2: Runtime of the stages in the prediction pipeline. Above each execution time column is the prediction time of the model in milliseconds. The size of each model is above each secondary column.

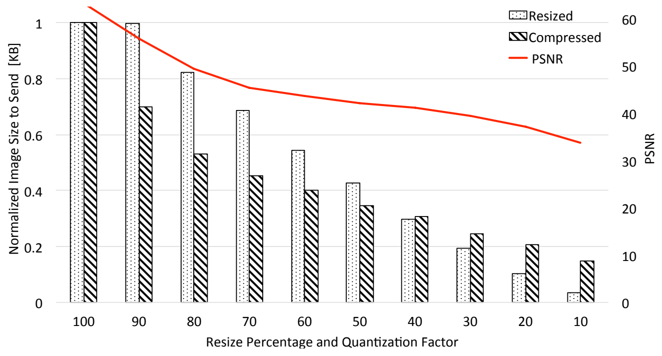


Fig. 3: Data-size sent of a compressed and resized image normalized to original image, and the effects of JPEG compression on PSNR.

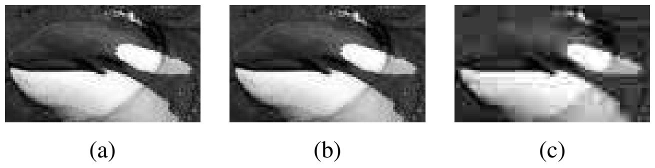


Fig. 4: JPEG compression: (a) Original image, (b) Q = 75, PSNR 39.9 dB, (c) Q = 10 PSNR 32.4 dB.

2. VISION PIPELINE

2.1. The Prediction Pipeline

We focus on feature extraction, description and matching, and prediction that have the potential to be executed onboard—prediction models are trained offline. Figure 2 suggests that most of the computation is at the beginning of the pipeline with prediction in most cases representing a small fraction. The benchmarks in this figure are the machine learning models we use to make predictions. We include the size of each model because they are trained offline and require onboard storage.

2.2. Reducing Image Size

We explore the effects of image compression or resizing on the prediction accuracy when applied before feature extraction. The JPEG compression algorithm [9] comprises 2 steps. First, dividing the image into 8x8 blocks of pixels and computing the frequency components using a DCT. Second, each frequency component is quantized according to a quantization factor (Q) which rounds low-occurring frequency components to zero, retaining essential information in the image. This is a lossy process where image information is lost but we show in our results that it does not significantly impact prediction accuracies. Figure 4 shows the differences in quality of two compressed images compared to the original.

Image resizing also reduces the amount of data to transfer by down-sampling the pixels in the image. We study the effects of JPEG compression and image resizing on prediction accuracies in our results. Figure 3 compares the amount of data transferred for JPEG compressed images varying the Q factor and resized images varying the resize percentage (results normalized to the original image). We also show the effects of varying Q on the PSNR (applies only to JPEG compressed images).

2.3. Network and Data Transfer

There are several scenarios where the mobile platform may be required to offload computation. For example, the case where onboard prediction would consume too much of the onboard resources to make a prediction in a reasonable time. To estimate the runtime trade-offs of onboard computation versus offloading, we add the uplink time to the total runtime. We study the following 3 scenarios: 1) prediction pipeline executed onboard, 2) preprocess the image and offload the pipeline, and 3) offload after feature extraction. We study the size of data transiting between stages using Google's protocol buffers [10]. Table 1 shows the size of the data after each stage in the pipeline for two different image sets. The Caltech data set is used to measure prediction accuracies whereas the Samsung Galaxy S3 images are much larger and have been used to show accurate runtimes in our final results.

Table 1: Payload Size

Image	Image Size (KB)	Feature Points (KB)	Descriptors (KB)
Galaxy S3	2,394.23	507.6	5,844.7
Caltech Dataset	62.6	66.7	784.5

3. PLATFORMS

In the mobile world, device temperatures must be cool enough for users to hold and must rely on small batteries. In most cases, the mobile device will also be used to make predictions on many different images using multiple classes. The devices are storage limited which means all the prediction models cannot be stored locally and must be retrieved from the cloud.

Mobile Platform with Accelerator is a Kayla mobile development kit with a Tegra 3 and an embedded NVIDIA GPU as our onboard accelerator. Figure 5 shows the potential of a GPU for feature extraction and description loads with speedups of 29× and 85×.

Network Base Station is the link between the mobile platform and the cloud; it incurs network constraints and delay.

The Cloud is assumed to processes all incoming requests without queuing delay. The configuration is summarized in Table 2.

4. METHODOLOGY

4.1. System Setup

The configuration for our setup is outlined in Table 2. As noted earlier we use a Kayla development board as our mobile platform.

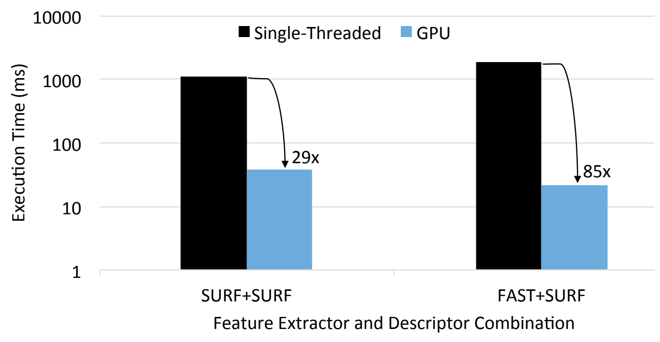


Fig. 5: Speedup of GPU vs Single-threaded. FAST is a feature extractor and SURF is a feature extractor and descriptor.

We use the newest OpenCV 2.4.6 [11] and CUDA 5.5 [12] versions which now support ARM platforms. Currently, OpenCV has few GPU implementations available of the benchmarks used.

4.2. Benchmarks

We use C++ implementations of the algorithms in the pipeline outlined in Figure 1 with training and prediction images from the Caltech 256 dataset [13] and a Samsung Galaxy S3 mobile phone. Since prediction is invariant of image size, prediction results are based on models trained on the Caltech dataset with approximately 100 images/class averaged across benchmarks. We use 80% of the images for training and the rest for validation. Final runtime results are based on images from the Galaxy S3 mobile phone. We use larger images from the S3 for the image-size dependent components of the pipeline (feature extraction, description, and matching) to give an accurate estimate of runtime.

4.2.1. Feature Extractors

We use common feature extractors including FAST, SIFT, SURF and ORB [14, 15, 16, 17]. Each of these feature extractors generate keypoints based on their implementation. FAST is a simple corner detector whereas SIFT, SURF and ORB are more robust to image differences such as scale, rotation, noise and illumination. We also use GPU implementations of FAST and SURF.

4.2.2. Feature Description and Matching

The keypoints are used to create descriptors for the image which cluster and describe a set of nearby keypoints. We use SIFT (single-threaded) and SURF (single-threaded and GPU) as the two descriptors in our pipeline. The descriptors are matched using a Brute-Force matcher and FLANN [18].

4.2.3. Prediction Models

The models (Normal Bayes Classifier, Boost, SVM, Gradient Based trees, Decision trees, and Regression trees) are trained in a 1-vs-all configuration using the Bag of Visual Words model (BoVW). The BoVW model extracts “words” of a class (such as the eyes, hands and head of a person) and creates a vocabulary to train the models. Using a histogram, the trained models search for occurrences of words describing the class in an image to make a prediction. Each class (positive sample) has a set of trained models to predict on the other classes (negative samples).

Table 2: Platform Configuration

Platform	Configuration
Mobile Platform	
OS	Ubuntu 12.04 Linux 3.1.10-carma armv7l
Processor	Quad-Core A9 (Tegra 3) @ 1.6GHz
Memory	2GB DDR3
L1 Cache	32KBi, 32KBD
L2 Cache	1MB
GPU	NVIDIA GeForce GT640/GDDR5
Cloud Server	
OS	Ubuntu 12.04.3 Linux 3.5.0-39-generic x86_64
Processor	8 $\times$ Intel Core i7-3770 CPU @ 3.40GHz
Memory	24GB DDR3
L1 Cache	256KB
L2 Cache	8MB

4.3. Measurements

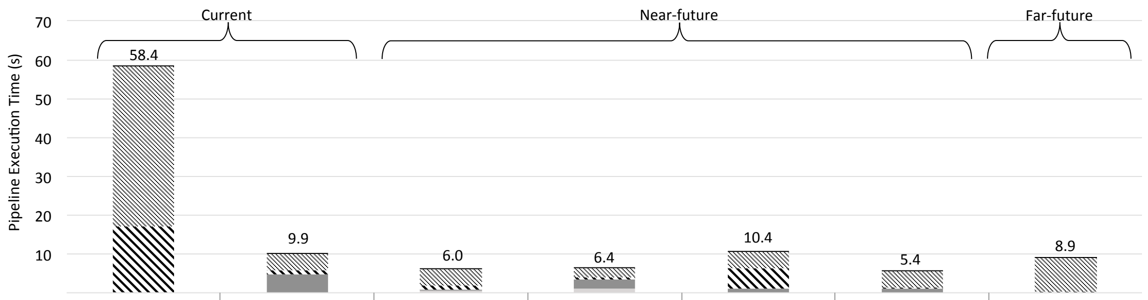
Our results focus on runtime, data-size, prediction accuracy and power. We measure the runtime in the pipeline using the *getTickCount()* function. The runtime budget is defined as a reasonable execution time to complete the pipeline. If one stage in a specific configuration is slower than the overall runtime, it exceeds our budget. This is the case for the FAST feature extractor on larger images. To model the size of transitioning data, we use Google protocol buffers, a binary format used for network serialization. We omit larger prediction models such as the NormalBayesClassifier because, when extrapolated to multiple classes, it would require GBs of storage. Our prediction accuracy is the number of correct predictions over the total predictions and we do not track false positives. Using [19], we measured the 4G LTE uplink bandwidth of the Verizon network to estimate network latencies in our results.

5. RESULTS

In this section we show where to execute the prediction pipeline under various constraints and the effects of image preprocessing on prediction accuracy.

5.1. Effects of Image Preprocessing

Figure 7 shows a clear tradeoff between runtime and prediction accuracy when down-sampling the image. As the image size is reduced and execution time decreases, prediction accuracy linearly decreases. Conversely, in Figure 8, the variations in image compression are small: the model accuracy varies by at most 2% compared to the original image. There is a slight increase in runtime at Q = 50 for the corner detector FAST because the quantization generates additional corners in its search window. Should the mobile device offload the prediction pipeline, we recommend to compress the image and send it over the network. We choose two competing schemes to reduce image size: one scheme with Q = 10 and another with a resizing percentage of 0.75 that have comparable prediction accuracies (within 5%). Even though the two configurations exhibit similar accuracies, we recommend compression because the amount of data to send over the network is much smaller (see Figure 6 Data Transferred entries).



	Onboard	Offload	Compressed at Q = 10	Resized at 0.75	Hybrid	Hybrid+GPU	GPU
■ Prediction (ms)	56	14	14	14	14	14	56
▨ Feature Description+Matching (ms)	41,317	4,067	4,067	2,434	4,067	4,067	8,464
▨ Feature Extraction (ms)	17,022	1,138	1,138	635	5,367	371	371
■ Upload (ms)	-	4,670	358	2,174	989	989	-
□ Compression/ Resize Time (ms)	-	-	437	1,130	-	-	-
Total Time (s)	58.4	9.9	**6.0**	6.4	10.4	**5.4**	8.9
Data Transferred (MB)	-	2.3	**0.2**	1.1	0.5	**0.5**	-
Total Energy (J)	93.4	3.5	**0.6**	2.2	8.7	**5.0**	116.5

Fig. 6: Breakdown of total runtime for all configurations. Runtime values based on Samsung Galaxy S3 images. The bold box shows a comparison of runtime, data transferred and energy for each configuration. The values for the recommended configurations are also in bold.

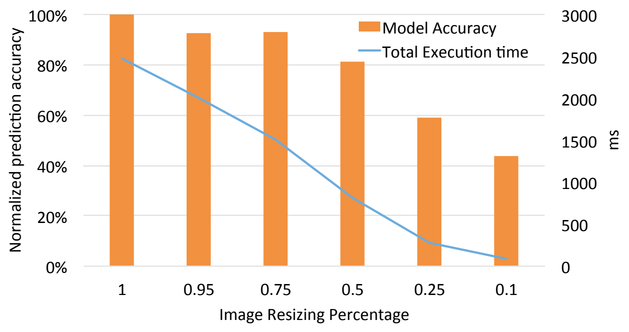


Fig. 7: Effects of image resizing on prediction accuracy. Runtime values based on the Caltech dataset images.

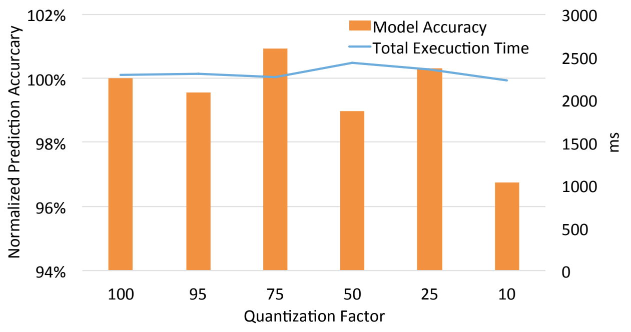


Fig. 8: Effects of JPEG compression on prediction accuracy. Runtime values based on the Caltech dataset images.

5.2. Offloading to the Cloud

The final results are presented in Figure 6 and are based on images taken with a Samsung Galaxy S3. We measure peak-power for each configuration and use LTE uplink power consumption measured in [5] to calculate the energy values. We do not include the download time from the server because it is negligible. The bold box shows a final comparison of runtime, data transfer and energy consumed for all configurations. Current configurations are either runtime or data upload intensive. Preprocessing the image achieves a reasonable runtime with low network usage. The time expelled compressing the image is compensated by offloading the rest of the pipeline instead of running it locally. As images taken on mobile devices grow in size, this network usage will increase. Although the hybrid approach is not energy efficient at the moment, in the future it will become more competitive because the size of feature data increases at a slower rate than image size. While a GPU-only approach without offloading achieves the lowest data transferred, we do not consider it a solution as it does not meet the power constraints of current mobile platforms.

5.3. Onboard Prediction

We also recommend to offload prediction because mobile device storage is limited. In the future when the entire pipeline can run locally, the mobile device can store many small models onboard and agglomerate results as a confidence value. Similarly, it can store several more robust and larger models (such as the Normal Bayes Classifier) and make a single prediction.

6. CONCLUSIONS

In this work, we performed the analysis of each stage needed in the image classification pipeline and proposed an efficient solution to maximize runtime and limit onboard resource usage. The best configuration for optimal runtime (11% faster) executes feature extraction with a GPU onboard and offloads the rest of the pipeline. Alternatively, compressing and sending the image over the network achieves lowest data transferred (2.5× better) and lowest energy usage (3.7× better) than the next best option.

7. REFERENCES

[1] Google, “Google glass,” http://www.google.com/glass/start/, accessed: 8 October 2013.

[2] J. Clemons, A. Jones, R. Perricone, S. Savarese, and T. Austin, “EFFEX: an embedded processor for computer vision based feature extraction,” in *Design Automation Conference (DAC), 2011 48th ACM/EDAC/IEEE*. IEEE, 2011, pp. 1020–1025.

[3] D.-U. Lee, H. Kim, M. Rahimi, D. Estrin, and J. D. Villasenor, “Energy-efficient image compression for resource-constrained platforms,” *Image Processing, IEEE Transactions on*, vol. 18, no. 9, pp. 2100–2113, 2009.

[4] C. Taylor and S. Dey, “Adaptive image compression for wireless multimedia communication,” in *IEEE International Conference on Communications, 2001. ICC 2001.*, 2001, vol. 6, pp. 1925–1929 vol.6.

[5] J. Huang, F. Qian, A. Gerber, Z. M. Mao, S. Sen, and O. Spatscheck, “A close examination of performance and power characteristics of 4G LTE networks,” in *Proceedings of the 10th international conference on Mobile systems, applications, and services*, New York, NY, USA, 2012, MobiSys ’12, pp. 225–238, ACM.

[6] M. S. Gordon, D. A. Jamshidi, S. Mahlke, Z. M. Mao, and X. Chen, “Comet: code offload by migrating execution transparently,” in *Proceedings of the 10th USENIX conference on Operating Systems Design and Implementation, OSDI*, 2012, vol. 12, pp. 93–106.

[7] E. Cuervo, A. Balasubramanian, D.-k. Cho, A. Wolman, S. Saroiu, R. Chandra, and P. Bahl, “MAUI: making smartphones last longer with code offload,” in *Proceedings of the 8th international conference on Mobile systems, applications, and services*, New York, NY, USA, 2010, MobiSys ’10, pp. 49–62, ACM.

[8] B.-G. Chun, S. Ihm, P. Maniatis, M. Naik, and A. Patti, “Faster and better: a machine learning approach to corner detection.,” in *Proceedings of the sixth conference on Computer systems*. ACM, 2010, vol. 32, pp. 105–19.

[9] G. K. Wallace, “The jpeg still picture compression standard,” *Commun. ACM*, vol. 34, no. 4, pp. 30–44, Apr. 1991.

[10] P. Buffers, “Googles data interchange format,” 2011.

[11] G. Bradski, “Dr. dobb’s journal of software tools,” 2000.

[12] C. Nvidia, “Compute unified device architecture programming guide,” 2007.

[13] G. Griffin, A. Holub, and P. Perona, “Caltech-256 object category dataset,” 2007.

[14] W. Förstner and E. Gülch, “A fast operator for detection and precise location of distinct points, corners and centres of circular features,” in *Proceedings ISPRS intercommission conference on fast processing of photogrammetric data*, 1987, pp. 281–305.

[15] D. G. Lowe, “Object recognition from local scale-invariant features,” in *Computer vision, 1999. Proceedings of the seventh IEEE international conference on*. Ieee, 1999, vol. 2, pp. 1150–1157.

[16] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-Up Robust Features (SURF),” *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, June 2008.

[17] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: an efficient alternative to sift or surf,” in *2011 IEEE International Conference on Computer Vision (ICCV)*. IEEE, 2011, pp. 2564–2571.

[18] M. Muja and D. G. Lowe, “Fast approximate nearest neighbors with automatic algorithm configuration,” in *International Conference on Computer Vision Theory and Application VISSAPP’09)*. 2009, pp. 331–340, INSTICC Press.

[19] T. N. LLC, “Testmy.net internet speed test v13,” Testmy.net, accessed: 1 October 2013.