Towards Real-Time Object Detection on Embedded Systems

Huizi Mao, Student Member, Song Yao, Student Member, Tianqi Tang, Student Member, Boxun Li, Student Member, Jun Yao, Member, and Yu Wang, Senior Member

Abstract—Convolutional neural network (CNN) based methods have achieved great success in image classification and object detection tasks. However, unlike the image classification task, object detection is much more computation-intensive and energy-consuming since a large number of possible object proposals need to be evaluated. Consequently, it is difficult for object detection methods to be integrated into embedded systems with limited computing resources and energy supply. In this paper, we propose a pipelined object detection implementation on the embedded platform. We present a comprehensive analysis of state-of-the-art object detection algorithms and select Fast R-CNN as a possible solution. Additional modifications on the Fast R-CNN method are made to fit the specific platform and achieve trade-off between speed and accuracy on embedded systems. Finally, a multi-stage pipelined implementation on the embedded CPU+GPU platform with duplicated module-parallelism is proposed to make full use of the limited computation resources. The proposed system is highly energy-efficient and close to real-time performance. In the first Low-Power Image Recognition Challenge (LPIRC), our system achieved the best result with mAP/Energy of 1.818e-2/(W·h) on the embedded Jetson TK1 CPU+GPU platform.

Index Terms—object detection, convolutional neural network (CNN), low-power, embedded system, pipeline

1 INTRODUCTION

The visual recognition problem is essential to computer vision research [1]. Thanks to the breakthrough in Image-Net Large-Scale Vision Recognition Challenge (ILSVRC) [2], Deep Neural Network (DNN), especially Convolutional Neural network (CNN), has led great advances in object classification tasks [3], [4]. The state-of-art performance of single-object recognition based on end-to-end neural network is steadily reaching or even exceeding performance of human visual system on the Image-Net dataset [5], [6]. Such a great success of deep learning on classification also influences other vision-related tasks like detection [2], segmentation [7], and caption [8].

Object detection, i.e. detecting the locations and categories of multiple object instances in one single image, is an essential problem in a large variety of applications such as robotics [9] and self-driving vehicles [10]. Recent advances in object detection are driven by the success of the Regions with CNN features (R-CNN) method [11]. By using the CNN pre-trained on large datasets for the image classification task and fine-tuned for the detection task, the R-CNN method greatly enhances the accuracy and outperforms the traditional object detection method with a 29% relative improvement [11].

For the R-CNN method, the improvement of accuracy involves great computation intensity. It usually takes tens of seconds to process a single image using R-CNN [11]. However, considering that a number of detection systems need to be integrated into mobile systems, such as smart phones, electronic glass and robots that are not attached to the power facilities, limited energy supply and computational resources could be a major problem in many real-life detection applications. In other words, energy conservation is of great concern for object detection on embedded systems.

Due to the concern of energy conservation, there has been growing interest in specific hardware design [7], [12], [13], [14], [15] for CNN and CNN-based vision tasks. Customized hardware, such as Application Specific Integrated Circuit (ASIC) and Field Programmable Gate Array (FPGA), has the potential to achieve both high performance and relatively low power consumption. In [14], Farabet et al. implemented an FPGA-based CNN detector. In [12], Chen et al. proposed the DianNao, which achieved high throughput and low energy consumption by introducing the tiling strategy and implementing dedicated buffer for data reuse. However, a comprehensive and flexible compiler is missing in DianNao, which makes it difficult to deploy rapidly-changing deep learning frameworks such as R-CNN, Fast R-CNN [16].

In contrast, implementing mainstream detection algorithms on commodity embedded hardware has been considered as an alternative solution. Commodity hardware like embedded GPU platform has a much higher energy efficiency compared to traditional desktop platforms with CPU and GPU [17]. Meanwhile it is much easier to develop flexible algorithms compared with customized hardware, since high-level programming languages like C++ and Python are well supported. However, most of the detection algorithms
are designed for desktop platform. Some critical problems like resources utilization and memory usage should be taken into consideration for compatibility and performance.

In this paper, we present an energy efficient implementation of object detection systems on an embedded CPU+GPU platform (Nvidia Jetson TK1 Board. The platform consists of a Tegra K1 SoC including a Kepler GPU with 192 CUDA cores and a 2.32 GHz ARM quad-core Cortex-A15 CPU [18]). To well utilize the heterogeneous architecture, we propose a pipelined object detection system and also introduce the modification in algorithm design. To be specific, this paper makes the following contributions.

- We introduce a comprehensive analysis of state-of-the-art CNN-based object detection algorithms. For the selected Fast R-CNN [16] algorithm, we propose additional modifications on it to fit the specific platform and achieve trade-off between speed and accuracy on embedded systems. For the newly state-of-art Faster R-CNN, we also give a system-level analysis of it.
- We propose a pipelined Fast R-CNN solution for embedded CPU+GPU platform to achieve both high speed and power efficiency. Our solution won the first prize and highest accuracy with low energy prize of the first Low Power Image Recognition Challenge (LPIRC). Based on the analysis of the bottleneck, we introduce the multi-stage pipeline and duplicated module-parallelism to further accelerate the detection.
- We performed design space exploration to analyze the selection of platforms and algorithms and the trade-off between accuracy and energy cost.

The rest of this paper is organized as follows. In Section 2, we give a brief introduction of the object detection task and standard benchmarks. In Section 3, we present a primer on CNN, give an analysis of detection algorithms, and propose our modification on Fast R-CNN algorithm for embedded systems. In Section 4, the pipelined implementations based on systematic analysis are proposed. In Section 5, we present our solution to train CNNs with different region proposal methods. Finally we report our experimental results on two perspectives of energy efficiency and accuracy in Section 6.

2 BACKGROUND

In this section, we give a formalized description of the object detection task. Besides, we also introduce two standard detection benchmarks and a recent competition which emphasizes both energy consumption and detection accuracy.

2.1 Object Detection Task

In the object detection task, algorithms are required to predict an arbitrary number (if any) of bounding boxes of arbitrary classes in an image, in which each bounding box is associated with an object category and a confidence value, as illustrated in Fig. 1.

The Intersection-of-Union (IoU) measure is used to evaluate whether a predicted bounding box matches the ground truth box. For two boxes A and B, the IoU is defined as

\[ \text{IoU} = \frac{|A \cap B|}{|A \cup B|} \]  

If a prediction is correctly labelled and its IoU with the ground truth box is no less than a threshold, this prediction is considered as a “positive example”. Otherwise it is a “negative example”. For example, in Fig. 1 each red box represents a “positive example” when the IoU threshold is set to 0.5.

To evaluate the overall accuracy of an object detection algorithm, four measures, recall, precision, average precision (AP), and mean average precision (mAP) are commonly adopted. The criteria are designed to penalize miss, duplication and wrong localization [2]. During test, for the i-th image and j-th prediction, the object detection algorithm is required to return a predicted bounding box \( b_{ij} \) with a category label and a confidence value \( c_{ij} \). Denoting the threshold of confidence by \( t \), \( s_{ij} = 1 \) if \( c_{ij} \geq t \), otherwise \( s_{ij} = 0 \). If the detection prediction \( j \) on image \( i \) matches a ground truth box with a confidence no less than the threshold \( t \), \( z_{ij} = 1 \). Otherwise, \( z_{ij} = 0 \). In this manner, all four measures can be defined.

Recall: For a given class, recall is the proportion of correct predictions over total objects in the images:

\[ \text{Recall}(t) = \frac{\sum_{i=1}^{N_i} \sum_{j=1}^{N_{ij}} z_{ij}}{N} \]  

where \( N_i \) is the number of images, \( N_{ij} \) is the number of total detections on image \( i \), and \( N \) is the number of total objects in \( s \) given class in all images.

Precision: Precision is defined as the fraction of correct predictions of total predictions. One groundtruth is counted only once and any duplicated detection would be marked as a false detection.

\[ \text{Precision}(t) = \frac{\sum_{i=1}^{N_i} \sum_{j=1}^{N_{ij}} z_{ij}}{\sum_{i=1}^{N_i} \sum_{j=1}^{N_{ij}} s_{ij}} \]  

Note that, \( s_{ij} = 1 \) when the algorithm determines that detection \( j \) in image \( i \) is an object in the given class, and \( z_{ij} \) measures whether it is a correct object in this class.
Average Precision: For multi-object detection, precision falls as the recall threshold rises. Therefore the metric Average Precision is proposed to measure the average performance Equation (4) represents the average precision on different recall levels. The average precision is defined as the integration of precision over the recall, where \( r \) stands for recall. Integration is the ideal case when the test dataset is infinitely large.

\[
AP = \int_0^1 \text{precision}(r)dr, \tag{4}
\]

In the practical case, average precision is often calculated with precision results on different recall levels. Assuming the difference between two close recall levels is \( d_r \), the average precision is defined as the average of precision results over different recall levels:

\[
AP = \frac{\sum_{\text{recall}=0.1/d_r,...,1} \text{precision}}{d_r + 1}. \tag{5}
\]

For example, \( d_r \) can be set to 1 [19].

Mean Average Precision: The mAP is adopted for object challenges as the final metric. Assuming there are \( N_o \) classes of objects in evaluated images, the mAP is defined as the average of AP over all classes:

\[
mAP = \frac{\sum_{n=1,2,...,N_o} \text{AP}_n}{N_o}. \tag{6}
\]

The mAP metric is able to evaluate an algorithm’s performance over all recall levels and all classes.

2.2 Datasets

There are two widely-used benchmarks for object detection method performance evaluation from two famous object detection challenges: the Pattern Analysis, Statistical Modeling and Computational Learning (PASCAL) Visual Object Classes (VOC) Challenge [19] and the Image-Net Large Scale Visual Recognition Challenge (ILSVRC) [2]. PASCAL VOC has been an annual event since 2006 with a public image dataset consisting of 20 classes obtained from the Flickr web site. ILSVRC releases a subset of Image-Net dataset which consists of millions of images and tens of thousands of classes in total. A 200-class subset of ImageNet is selected for the ILSVRC detection task. All images for the ILSVRC detection task are manually labelled with axis-aligned bounding boxes indicating the position and category of the corresponding instance. The benchmarks provided by PASCAL VOC and ILSVRC have become two standard benchmarks for the large-scale object detection research.

The PASCAL VOC 2007, 2010, and 2012 datasets are frequently adopted in object detection research. For training and validation, the three datasets provide 9963 images, 10103 images, and 11530 images with objects in 20 classes respectively. The ILSVRC2014 detection dataset consists of much more images compared with the PASCAL VOC dataset: There are 456567 images for training, 20121 images for validation, and 40152 images for testing. Due to their different sizes and characteristics, one method usually achieves different results on two benchmarks. For example, R-CNN [11] achieves the mAP of 58.5% on VOC 2007 test dataset and 31.4% on ILSVRC 2013 test dataset [16]. DeepID-Net [20] (current state-of-art method) achieves the mAP of 64.1% on VOC 2007 test dataset and 50.3% on ILSVRC 2014 test dataset [20].

2.3 Low-Power Image Recognition Challenge (LPIRC)

The Low-Power Image Recognition Challenge (LPIRC) is a competition that aims to boost the development of technologies in both detection performance and energy conservation. In this challenge, not only the output of detected bounding boxes, but also the power consumption will be recorded. The results will be evaluated based on both recognition accuracy and energy consumption, as illustrated in Eq. 7, where \( E \) is the total power consumption during the test.

\[
\text{Score} = \frac{mAP}{E} \tag{7}
\]

For LPIRC, the training dataset and validation dataset come from the ILSVRC 2014 detection task. The test dataset, which consists of 5000 images, was derived from ImageNet with some new images. During the competition, the detection system is required to process 5000 images within 10 minutes. If the detection system fails to process all 5000 images, it will be regard as predicting no bounding box in the unprocessed images. In this manner, the mAP results in LPIRC may be far lower than the theoretical value if the detection system only process a small set of test images.

3 ALGORITHM DESIGN

In this section, we introduce how to map object detection methods on embedded systems from the perspective of algorithm. First, a primer on CNN is presented. After that, we analyze the pros and cons of different state-of-the-art detection algorithms including R-CNN, SPP-net, Fast R-CNN, and Faster R-CNN, and choose the most suitable one for our implementation. The specific modifications of the selected Fast R-CNN algorithm for embedded CPU+GPU system are introduced at last.

3.1 Primer on CNN

CNN [21] achieves state-of-the-art performance among a large number of artificial intelligence applications in recent years. A typical CNN consists of a number of layers that run in sequence. The intermediate results between different layers are named feature maps. A three-layer CNN example is shown in Fig. 2 (a). It should be noted, there can be a large number of feature maps rather than one feature map between two layers.

Among all types of layers, four layer types are commonly used in CNN, including the convolutional layer (CONV layer), the fully-connection layer (FC layer), the pooling layer, and the nonlinear layer.

CONV layer: A CONV layer has multiple local filters (also known as “kernel”) that act as convolution kernels to the input feature maps. Denoting the kernel size by \( K_x \times K_y \) for each \( K_x \times K_y \) window of the input feature map, the kernel generates an output value by summing the product of each weight in the kernel and the value at the corresponding position in the window of the input feature
Fig. 2: Examples of CNN: (a) The general structure of a simple CNN; (b) A convolutional layer example with one input/output feature map and the pseudo-code; (c) A max pooling layer example with one input/output feature map and the pseudo-code.

An example of CONV layer that connected to one input feature map and one output feature map is shown in Fig. 2(b) where the kernel size is $2 \times 2$. The pseudo-code of the CONV layer is also presented in Fig. 2(b). If a CONV layers have multiple kernels, there will be multiple output feature maps.

**Pooling layer:** A pooling layer is able to extract the local information and reduce the size of feature maps. Denoting the pooling kernel size by $K_x \times K_y$, each value in the output feature map is pooled from a $K_x \times K_y$ window in the input feature map. Two types of pooling functions are commonly used, max pooling and average pooling. As shown in Fig. 2(c), assuming the pooling layer is connected to one input feature map and one output feature map, a $2 \times 2$ max pooling kernel extracts the maximum value from a $2 \times 2$ region on feature map 1 to feature map 2 and discards other values. The pseudo-code of the max pooling layer is also presented in Fig. 2(c). With the increase of pooling kernels, the number of output feature maps increase.

**FC layer:** An FC layer applies affine transformation from the input feature map to the output feature map. The input feature map is regarded as an $1$-d vector denoted by $x$ of length $N_x$, and the output feature map is a vector denoted by $y$ of length $N_y$. In this manner, the affine transformation can be expressed with a matrix $W$:

$$y = W \times x + b,$$

(8)

where $b$ is a bias vector.

**Nonlinear layer:** Non-linearity is essential to neural network’s representation capability [22]. Introducing nonlinearity into CNN can significantly improve the performance of CNN on complex tasks. In CNN, nonlinear layers are usually introduced after CONV layers, pooling layers, and FC layers. In early works, $f(x) = \tanh(x)$ or $f(x) = (1 + e^{-x})^{-1}$ were often used to model a neuron’s output. Recently, the Rectified Linear Unit (ReLU)

$$f(x) = \max(0, x)$$

(9)

is often used as the nonlinear function in the nonlinear layers.

It should be noted, the size of one layer’s input feature maps should be identical to the size of its previous layer’s output. The CONV layer is able to adapt to different input scale due to the intrinsic characteristics of convolution, but the FC layer can only take fixed-size input. Therefore any CNN model with FC layers should process images with fixed scale (e.g. $224 \times 224$ [23]), else warping or cropping is necessary. Such a limitation is advisory for the detection of arbitrary-scale objects. Variations of CNN are recently proposed to deal with this problem [16, 24], which will be illustrated in the following subsection.

To use a CNN for image classification, object detection, or any other task, one should **train** the CNN on a dataset. Recent researches [25] showed that, a CNN trained on a large dataset for a given task can be used for other tasks and achieved high accuracy with minor adjustment in network weights. This minor adjustment is called “fine-tune”.

### 3.2 Algorithms Selection

R-CNN based object detection systems have three main steps: 1) coarse-grained region proposal generation; 2) CNN feature extraction and object classification; and 3) fine-grained bounding box compression and regression. Fig. 3 briefly illustrates these three steps. The region proposal extraction algorithms, based on some priors of object characters such as texture and contour, predict the bounding boxes of possible objects. A CNN that is fine-tuned for detection task together with a classifier such as a linear SVM predicts the category and confidence value of the object in each bounding box. After that all bounding boxes are refined to give final results. Bounding box regression is an optional step to calibrate the coordinates of output bounding boxes.

The breakthrough in object detection achieved by R-CNN has pointed out two promising directions to further improve object detection performance. One is to further increase accuracy by introducing more stages, more sophisticated systems and larger networks, e.g. the Deep-ID Net framework proposed by Ouyang et al [20]. It splits background
rejection and object rejection into two stages, adds one more stage named Contextual Modeling to refine the confidence scores and averages the results of multiple models, which achieves the mAP of 50.3% on ILSVRC 2014 test dataset. The other direction is to speed up detection by fusing multiple stages together and avoiding redundant computations, e.g., the work of Spatial Pyramid Net (SPP-net) [24], Fast R-CNN [16] and Faster R-CNN [26]. We generally focus on the speedup methods, since they are able to improve the performance with orders of magnitude. A summary of three speedup methods and the R-CNN baseline are shown in Fig. 4.

The drawback of R-CNN is obvious: during test time, the whole network performs feed-forward for each region proposal, making it rather computation-intensive. Denoting the time of region proposal methods by \( t_{rp} \), the time on CONV layers by \( t_{CONV} \), and the time on FC layers by \( t_{FC} \), the total runtime of R-CNN can be estimated as

\[
\begin{align*}
    t_{R-CNN} &\approx t_{rp} + t_{CONV} + t_{FC}, \\
    t_{CONV} &\propto N_r \cdot S_x \cdot S_y, \\
    t_{FC} &\propto N_r \cdot N_l^2,
\end{align*}
\]

where \( N_r \) is the number of region proposals, \( S_x \times S_y \) is the size of the input image, and \( N_l \) is the length of feature vectors that are connected to FC layers. The \( t_{CONV} \) is in proportion to the size of the input image. This is because the kernel size and the corresponding sliding window size of input feature maps is static, thus a larger image leads to a larger sliding window number and convolution operations.

The convolution operations are highly time-expensive. Since in R-CNN there are on average \( N_r = 2000 \) proposals for each image, the whole CNN is executed for 2000 times for each image, thus the runtime of R-CNN is extremely long. Even on a Nvidia K40 GPU, it takes 47 seconds to process one single image with VGG-16 Net [11], which is far from the real-time performance. SPP-net, Fast R-CNN and Faster R-CNN are all variations of R-CNN which aim to overcome R-CNN’s speed problem. SPP-net achieves a speedup of \( 10 \times \) compared with R-CNN by reusing the feature map after all CONV layers [24] and thereby avoids repeating the expensive convolutional operations. Fast R-CNN [16] uses a simplified Spatial Pyramid Pooling in SPP-net, named RoI pooling, and combines classification and bounding box regression with a single CNN. There are only minor differences between the feature map reusing methods of Fast R-CNN and SPP-net, i.e., the Spatial Pyramid Pooling of SPP-net and the RoI pooling of Fast R-CNN. Fast R-CNN outperforms SPP-net on PASCAL VOC with a small margin (66.9% versus 63.1%). Since Fast R-CNN is faster and more accurate than SPP-net, it is more preferable for our implementation. The runtime of SPP-net and Fast-RCNN can be estimated as

\[
\begin{align*}
    t_{Fast-R-CNN} &\approx t_{rp} + t_{CONV}' + t_{FC}', \\
    t_{CONV}' &\propto S_x \cdot S_y, \\
    t_{FC}' &\propto 2 \cdot N_r \cdot d \cdot N_l,
\end{align*}
\]

where \( d \) is the dimension of the diagonal matrix \( S_d \) in truncated Singular Value Decomposition (SVD):

\[
W \approx U_d \cdot S_d \cdot V_d = U_d' \cdot V_d'.
\]
Fig. 5: Fast R-CNN v.s. Faster R-CNN. Region Proposal module is replaced with a small Region Proposal Network (RPN), with feature maps reuse.

Since the CONV layers are executed only once when analyzing multiple region proposals in one image, the time cost on CONV layers is reduced dramatically, i.e., \( t_{\text{CONV}} \ll t_{\text{CONV}} \). Besides, as usually \( d \ll L, t_{\text{FC}} \) is also significantly reduced.

For Fast R-CNN, the region proposal stage remains as the bottleneck. This is because traditional region proposal extraction methods are implemented on CPU platform. For example, in our implementation with Fast R-CNN, region proposal extraction with Edge-Boxes [27] composes 52.4% of total runtime.

Faster R-CNN is the state-of-art detection solution so far. Faster R-CNN [26] obtains great speedup compared with the other two methods as illustrated in Fig. 5. The time-consuming region proposal module running on CPU is replaced with a small Region Proposal Network (RPN) running on GPU. With RPN, the runtime of Faster R-CNN can be estimated as

\[
t_{\text{Faster R-CNN}} \approx t'_{\text{CONV}} + t_{\text{FC}}. \quad (17)
\]

We give a system-level analysis at Section 4.5. According to our analysis, Faster R-CNN on Tegra K1 would not have advantage over speed, but would have higher accuracy and lower power consumption. Because it was proposed just several days before the LPIRC competition, we did not implement it for the competition.

### 3.3 Modifications on Fast R-CNN

As shown in Eq. 14 and Eq. 15, with a given CNN, the time consumed by CONV and FC layers is mainly determined by the input image scale and the parameter selection in SVD. However, aggressive parameter selection to reduce the runtime may also significantly degrade the detection accuracy. In this subsection, we first introduce the kernel part of Fast R-CNN, the Region-Of-Interest (RoI) pooling. After that, modifications on Fast R-CNN with scale selection and SVD are introduced. In addition, we investigate different region proposal methods to match the speed of CNN.

#### 3.3.1 RoI Pooling

Fast R-CNN reuses the features extracted by CONV layers using RoI pooling to avoid intensive and redundant computations. Fig. 6 illustrates this method. During RoI pooling, the regions of interest in the original image are projected on the feature maps and then are pooled into fixed number of bins by similar method of Spatial Pyramid Pooling method (SPP) [24]. The size of each bin is adaptively set to ensure that the output size fits the input of FC layers and meanwhile maintain the essential information needed for recognition. In this way, the CONV layers, which are time-expensive, perform only one single feed-forward for each image and the inference can be significantly speeded up.

For various projection area sizes, the RoI pooling can generate a fixed-length feature vector, thus Fast R-CNN can directly process images in different scales. However, it should be noticed that, the size of feature maps is reduced when passing through each CONV layer. For example, CaffeNet [23] reads images in 224×224 pixels, but the size of the feature map after the last CONV layer is only 13×13. In this manner, if the feature map after the last CONV layer is too small, it will be hard to detect small objects. Practically, to ensure that small objects are detected with CaffeNet, the resolution of input images should be larger than 224×224.

#### 3.3.2 Parameter Selection

As shown in Eq. 13, the runtime of CONV layers is in proportion to the input image size, i.e. the square of the input image scale. Besides, smaller scales can also reduce the amount of memory required for intermediate data. However, typically it is easier to detect small objects in a large-scale image. Moreover, if the input image scale is too small, the detection accuracy may fall drastically as a large number of false negatives and false positives of small objects may be introduced. An appropriate range of input scales should be determined to achieve both speedup and low accuracy degradation. Defining the scale \( s \) of an image as the length of its shortest side, we set \( s \) to 600 in our Edge-Boxes-based solution and 450 in our BING-based solution according our experiments. The largest side is also restricted to 1000 and 750, respectively.

For image classification where both CONV layers and FC layers are executed once, the runtime of FC layers can be omitted compared with the runtime of CONV layers.
However, for Fast R-CNN, since FC layers are executed for hundreds of times or thousands of times while CONV layers are executed only once, the runtime of FC layers cannot be ignored.

As shown in Eq. [12] and Eq. [15], speedup can be achieved by employing SVD on FC layers. Denoting the exact length of the feature vector before and after a FC layer as $M$ and $N$, the affine transformation in the FC layer can be expressed with a $M \times N$ weight matrix $W$, as mentioned in Eq. [8]. The SVD decomposes the original $M \times N$ weight matrix $W$ into the product of two matrices of size $M \times d$ and $d \times N$. In this manner, the total amount of computations of the FC layer (measured by number of multiplications) is reduced from $M \times N$ to $d \times (M + N)$. When $d \ll M$ and $d \ll N$, the runtime of FC layers can be significantly reduced.

Though SVD has the potential to greatly speedup FC layers, if the parameter $d$ is too small, a lot of information in the original matrix $W$ is discarded and the object accuracy may fall sharply. Consequently, it is important to select the suitable parameter $d$ in SVD to reduce the total computations while maintain a comparable accuracy to the original FC layers. In our BING-based implementation, the first two FC layers (9216 $\times$ 4096 and 4096 $\times$ 4096) are decomposed and the top 1024 singular values are selected. Compared with the original FC layers without SVD, the employed decomposition reduces the total computations from 55 million operations to 26 million operations in a single run. When the number of proposals is set to 200, the modified CNN with SVD leads to a 26% reduction of the total runtime compared with the original network.

### 3.3.3 Selection of Region Proposal Methods

Region proposal methods aim to improve detection accuracy and avoid meaningless exhaustive computation by focusing on the most salient parts of an image. Compared to naive methods like uniform sampling and sliding-window method, region proposal methods can greatly reduce the overall computations, thus enabling the use of expensive classifiers per window [11]. Two general approaches for region proposal generation are grouping methods and window scoring methods. Grouping methods start with low-level segmentations and then merge them based on high-level features. According to the survey of [28], most grouping methods are relatively more computation-intensive. A faster alternative is to score all sliding windows directly and select windows with highest scores. Typically this approach generates proposals with lower recall. Edge-Boxes and BING both fall into the latter category.

A recent survey [28] of region proposal methods on desktop or server PC platform provides us with a qualitative estimation of all algorithms’ speed and performance on embedded platform. The speed and performance results of four widely used region proposal extraction methods including Binary Normalized Gradient (BING) [29], Edge-Boxes [27], Selective Search [30], and Geodesic Object Proposal (GOP) [31] are listed in Table 1.

Among all available methods, an ideal choice is to choose the region proposal method that operates at same speed to CNN classification. In this manner, a two-stage pipeline can reduce the runtime by 50%. Since forward pass of VGG-16 net in Fast R-CNN takes 223ms [16] on PC platform with a Tesla K40 GPU, BING and Edge-Boxes which take 0.2s and 0.3s to generate region proposals are more favorable for the detection system. Though Selective Search and GOP are also two commonly used methods in detection, since their speed is far below satisfactory even on the PC platform. They are not good choices for detection pipeline.

Edge-Boxes and BING are both efficient algorithms. Edge-Boxes introduces a bottom-up method which first detects the contour and then measures spatial relationship of a box and the contour of it to judge the possibility of an object in the box. BING rescales the image into different sizes and uses the sliding windows to measure the possibility (which is defined as the objectness feature) of an object in an area of every rescaled image. With the rescaling step, BING is much more efficient than the original sliding window method for generating region proposals.

### 4 System Implementation

In this section, An overview of the detection system is presented. We then introduce our two-stage pipelined detection system design that parallelizes region proposal and CNN. Based on it, two variant solutions are further presented to solve the bottlenecks of EdgeBox algorithm and network communication. Compared with the original two-
Table 2: A brief summary of different module features

<table>
<thead>
<tr>
<th>Module</th>
<th>Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Uploading/Downloading</td>
<td>Bandwidth consuming</td>
</tr>
<tr>
<td>Region proposal</td>
<td>Computation-intensive Logic-complex</td>
</tr>
<tr>
<td>CNN classifier</td>
<td>Greatly computation-intensive</td>
</tr>
<tr>
<td>NMS</td>
<td>Logic-complex</td>
</tr>
</tbody>
</table>

4.2 Implementation Details of Specific Modules

Region proposal methods on ARM CPU. Edge-Boxes[4,5] and BING[6,7] are both available as open-source codes on GitHub. Edge-Boxes are written in matlab codes and recompiled into C++ codes with the Mex tool.[8] We deploy the original codes with a pure C++ implementation on the ARM quad-core Cortex-A15 CPU of the Jetson TK1 platform for the consideration of simplicity, compatibility, and performance.

Edge-Boxes and BING both exploit the SSE (Streaming SIMD Extensions) to accelerate the algorithms with instruction-level parallelism. However, SSE is supported only on x86 platform. We replace SSE with equivalent ARM NEON intrinsic APIs to achieve similar performance.

CNN models for different implementations. We choose the Berkeley Vision and Learning Center (BVLC) version CaffeNet for the CNN part in our implementation. CaffeNet is based on AlexNet proposed by Krizhevsky et al. [23] with minor modifications. CaffeNet is a “light-weight” model compared with other networks for ImageNet dataset, which consists of five CONV layers and three FC layers. An additional FC layer, named as bounding box regression layer, is added to the selected CaffeNet model as a sibling output layer to calibrate the bounding box coordinates. In our BING-based system, we eliminate the bounding box regression layer and perform SVD decomposition on FC layers due to consideration about the speed.

4.3 Pipeline Design

As mentioned above, the overall performance of the detection system is determined mainly by region proposal generation and execution of CNN. Besides, the execution of CNN is convenient to deploy on GPU while region proposal extraction methods are not. Since the two major tasks demand for different computation platform, it is natural to design a pipeline system to achieve parallelism and maximize the utilization of resources on-board, especially the four CPU cores and the GPU.

4.3.1 Pipeline Stage Design

Our primary experiments with the Edge-Boxes-based system indicate that Edge-Boxes algorithm and CNN classifier take up most of runtime. Therefore we design a two-stage pipeline scheme, as illustrated on the left of Fig. 9. A queue is inserted into the mid of the region proposing module and the CNN classifier as a buffer to enable parallel proposal generation and CNN classification, as well as balance the possible time fluctuation of the two sides due to different characters of images. A unit of the queue is less than 1KB, as we limit the number of proposals to 200, which imposes negligible time overhead.

4.3.2 Duplicated Modules

In our first Edge-Boxes-based implementation, the region proposal generation module is the bottleneck of the system. As Jetson TK1 platform has four ARM cores on-board, multi-threading of Edge-Boxes algorithm seems favorable.

References:
1. https://github.com/pdollar/edges
Therefore we implement a system with two module running Edge-Boxes algorithm simultaneously, writing results to the same queue, as illustrated in the right of Fig. 9. With duplicated modules, the throughput of the first stage is greatly increased and thereby the overall detecting process is accelerated. Our results demonstrate that a relative 50% speedup can be achieved with duplicated modules compared to the original Edge-Boxes-based implementation.

According to our experiments, a single Edge-boxes module costs 260MB memory while a BING module costs only 8MB. The neural network module typically costs 700-800MB memory usage depending on the size of input image. The overall memory usage of different systems are listed in Table. 3. Notice that the system takes about 160MB of memory. Considering the overall available memory on Tegra K1 is about 1900MB, the two-threaded Edge-boxes solution aggravate the risk of memory failure. During the competition, we limit the max size of image to 1000 pixels to avoid memory failure.

**TABLE 3: Memory usage of different systems**

<table>
<thead>
<tr>
<th>System</th>
<th>Memory Usage(MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Edge-boxes, pipeline</td>
<td>980-1230</td>
</tr>
<tr>
<td>Edge-boxes, two threads</td>
<td>1240-1330</td>
</tr>
<tr>
<td>BING, two-stage pipeline</td>
<td>900-930</td>
</tr>
<tr>
<td>BING, three-stage pipeline</td>
<td>940-970</td>
</tr>
</tbody>
</table>

4.3.3 Additional optimizations

In the LPIRC competition, we found that the network interface bandwidth became the bottleneck of our BING detector when the detection speed was fast enough. To deal with this problem, an option is to divide the system into more stages as shown in the middle of Fig. 9. Though the last two modules, NMS and uploading module, cost less amount of computations, their speed is limited by network bandwidth. In this case, another queue is inserted into the mid of the classifier and the NMS module to buffer the data. Our further experiments demonstrate a relative 43% speedup over the original two-stage BING-based system.

Another method to improve the latency is to compute the convolutional layers simultaneously with the region proposal method, just as shown in Figure 8. In the sense of latency, it helps as the critical path for the first image is changed from “Region Proposal – CONV – FC” to “Region Proposal(CONV) – FC”. However, it helps little with the overall throughput. The overall throughput depends mainly on the bottlenecks, which in this case both are the Region Proposal module. It may help in some extremely latency-sensitive cases such as self-driving.

4.4 Exception Handling

Due to the existence of some special images in Image-Net dataset and possible prediction errors, the system should be robust enough to handle common exceptions. We have met and handled the following exceptions: file corruption; gray-scale images; images with no proposal generated; super-large-scale images that cause memory error in Edge-Boxes module. Images with any of the above features are passed during competition. During the competition we limit the scale of an image to 1000 and set a timeout limit of Edge-Boxes module to 3 seconds.

4.5 Analysis of Faster R-CNN

On desktop, Faster R-CNN’s speed advantage mainly comes from replacing region proposal with RPN, as shown in Figure 5. However, it is noticed in Table 4 that, Edge-Boxes on Tegra K1 takes 60% of total time, while it takes 80% on the desktop. As they are near-balanced on CPU and GPU, region proposal and CNN can be parallelized, hence greatly increases the overall throughput. That implies Faster R-CNN may not have such large advantage on Tegra K1.

On the other hand, RPN runs fully on GPU and takes additional 0.13s on Tegra K1. It leads to a total runtime of 1.08s. Compared with our duplicated-modules two-stage pipeline(1.06s), Faster R-CNN may not have advantage in speed. Still, Faster R-CNN has lower power consumption(due to lower CPU workloads) and higher mAP [26] than Fast R-CNN.

5 TRAINING DETAILS

The training procedure is performed upon the framework of R-CNN and Fast R-CNN with minor modifications to fit different proposal methods and additional data of ILSVRC2014. In the training dataset, images are NOT exhaustively annotated. Therefore, there are questions about...
how to effectively train the detection models. In R-CNN [11], the author chooses to rely heavily on the validation set. Following a strategy similar to R-CNN, we split the validation set into $val_1$ and $val_2$ with balanced distribution of every object class. During training, $val_1$ plus the auxiliary ILSVRC2014 training dataset is used. Unlike what Girshick et al. [11] did, we do not use the ILSVRC2013 training dataset because we notice that the ILSVRC2014 dataset is better annotated and as well large enough for training. Also, we discard all the images without any object in ground-truth to improve the quality of our training data.

We start with the CaffeNet model [32] and add a bounding-box regression layer to it. In order to prepare more training data, both Selective Search and Edge-Boxes methods were used to generate region proposals (4k proposals per image in total). It is observed that training with proposals generated by both proposal methods provides better results than training with any single method alone. In the first stage, the object proposals provided by Edge-Boxes and Selective Search are exploited to train the model with a larger learning rate (we set the initial learning rate $10^{-3}$). The model of this stage will be used for the Edge-Boxes detector. It is noticed that, training with both two proposal sets provides better results than that with any one set (see Table 7 in Section 6 for details). We conjecture that the selective search proposals help avoiding the model from over-fitting.

For Edge-Boxes-based detection system, the CNN model is further fine-tuned with the proposals provided by BING. We do not fine-tune the CNN model with proposals provided by BING directly due to the unsatisfying quality of proposals generated by BING. It may cause the deterioration of detection accuracy. To distinguish with the concept of domain-specific fine-tune proposed in R-CNN [11], we name this procedure task-specific fine-tune, which implies that neural network has capability to adapt to different front-end methods.

6 EXPERIMENTAL RESULTS

In this section, we first present our results in LPIRC. During the competition, all data are transferred through LAN. After that, an analysis based on local experiments is given. The effectiveness of the proposed pipelined detection systems is evaluated by comparing with other platforms and exploring the design space. The influences of different training methods are also investigated.

In the experiments, two CNN models pre-trained with ILSVRC dataset are used, including the original CaffeNet in Edge-Boxes-based implementation and CaffeNet with truncated SVD in BING-based implementation. This is because the Edge-Boxes-based solution is designed to optimize accuracy while the BING-based solution is designed specifically for high speed. The two models are trained as described in Section 5. Offline experiments are performed with images pre-stored on the internal flash memory of Jetson TK1 platform. Online experiments are performed with a server connected via 100Mbps Ethernet.

6.1 Results in LPIRC

In LPIRC, we proposed two solutions including a BING-based one and an Edge-Boxes-based one. The BING-based solution achieved the mAP of 2.971e-2 (over 5000 images) and the energy cost of 1.634W/h . The Edge-Boxes-based solution achieves the mAP of 1.816e-2 and the energy cost of 1.574 Watt-Hour. The final score of mAP/energy for BING-based and Edge-Boxes-based solutions are 1.818e-2 and 1.154e-2, respectively. The two solutions won the first prize, the third prize and the Highest Accuracy with Low Energy prize.

It should be noted, as mentioned in Section 2.3, only 10 minutes were given for the detection systems to process images. In this case, the proposed systems only processed hundreds of images, which were only a small set of the 5000-image test dataset. However, the mAP results were calculated over all 5000 images. Consequently, the mAP results in LPIRC are far lower than theoretical values.

With our further experiments, the Edge-Boxes-based system is proved to achieve a higher accuracy compared with the BING-based system. However, the advantage of Edge-Boxes-based systems on accuracy is not enough to compensate the influence of its relatively lower speed. Consequently, the BING-based solution achieved a better result in LPIRC by processing more images.

6.2 Analysis of Parameter Selection

Selection of proposal number. Number of generated region proposals determines the total amount of computations of the CNN classifier. As illustrated in Fig. 10 the classifier’s runtime increases almost linearly with the input number of proposals. The intercept and the slope corresponding to the one-frame time cost of CONV layers. The slope stands for one-frame time cost of FC layers.

![Fig. 10: Running time of the CNN in detection pipeline with different region proposal number. Reduced stands for the pruned network for BING-based implementation; Original stands for the network adopted in Edge-Boxes-based implementation. Time cost increases almost linearly with the number of proposals. The intercept on y-axis stands for time cost of CONV layers. The slope stands for one-frame time cost of FC layers.](image-url)

In [16], the relationship between accuracy and the region proposal number is analyzed. However, the analysis in [16] does not cover the situation when the proposal number is below 1000. For real-time detection, processing more than 1000 proposals would be quite computation-expensive. As illustrated in Fig. 10 the runtime of the CNN doubles when the number of proposals increases from 200 to 1000, while mAP increases by merely 1.4% as shown in Fig. 11. Considering that, in our implementation, for each image, 200...
region proposals are extracted to achieve a high speed while introducing no significant accuracy drop. Consequently, we choose an empirical number 200 as the number of proposals.

**Selection of image scales.** In [16], the author analyzed the trade-off between multi-scale detection and single-scale detection. We show that in single-scale detection, the scale size can greatly influence the final performance. As the accuracy of a wisely-chosen scale can achieve similar results with multi-scale detection (less than 2% loss of mAP [16]), we mainly focus on the running time of CNN module as shown in Fig. 12. According to the curves, different scale ranges are selected for Edge-Boxes-based system and BING-based system.

### 6.3 Analysis of System Implementation

**Comparison of platforms.** The energy efficiency of neural network is of prior concern as it exists in almost all detection algorithms. Here we tested the speed and power consumption of AlexNet on four platforms: Tegra K1 GPU, ARM Cortex A-15, Intel Xeon E5-2690, Tesla K40. Other embedded platforms, like Intel Atom or AMD G-Series processors, has been demonstrated less energy efficient than ARM [33]. The power of Xeon E5-2690 is measured with Intel Performance Counter Monitor. The power of Tesla K40 is measured with nvidia-smi utilities. According to Table 5, the embedded GPU has the highest energy efficiency.

We give a rough comparison of Jetson Tegra K1 platform and PC platform regarding the performance of different modules in Table 4. The runtime of Edge-Boxes and BING on PC comes from third-party results provided by Hosang et al. [28]. In the original paper of Edge-Boxes [27], the runtime is given as 0.25s. In that of BING [29], the runtime is given as 0.003s, however, such a speed is obtained by excessive approximations [35], thus not adopted here. The experiment of CaffeNet on PC platform is performed with a Tesla K40. Due to the diversity of PC configuration and fluctuation of energy consumption, we only give an approximate estimation of overall power. The power of PC is underestimated as 150W, which is merely the sum of the net power of Tesla K40 and Intel Xeon E5-2690. The power of Tegra K1 is measured with a power meter during the runtime. This comparison shows that Jetson TK1 has advantage over the desktop in energy efficiency, no matter CPU or GPU is the bottleneck of the whole system.

As FPGA provides higher energy efficiency compared to GPU, here we compare the performance of the state-of-art FPGA accelerator and mobile GPU. To the best of our knowledge, we have not found any FPGA platform that is able to run fast or faster R-CNN. So we simply compare the power efficiency of convolutional layers between Tegra K1 and the recent state-of-art FPGA solution [36]. The power efficiency of Tegra K1 is 2.45 GOPs/W/s, while that of FPGA is 19.5 GOPs/W/s [36]. In the sense of power efficiency, FPGA outperforms mobile GPU by a large margin. However, the major obstacles of are FPGA includes:

- Thorough study of accuracy-precision relationship of object detection algorithms. Most FPGA platforms only support low precision algebra, or higher precision will lead to little advantage over mobile GPU solution [15], [36].
- More support for special layers like ROI pooling. Otherwise mobile CPU has to do it, which is rather energy inefficient.
- An easier interface is needed to build a flexible system.

Due to these unsolved problems, mobile GPU is considered as a more preferred choice.

**Analysis of module-wise running time.** We perform module-level time/energy analysis under different circumstances as illustrated in Fig. 13. The detection procedure is set in the same way as in LPIRC. The final running times of different modules are averaged over 500 images randomly sampled from the validation dataset. The time cost of network interface, though varying by the network conditions, provides a qualitative comparison to other modules. It is no-
In this paper, we propose an energy efficient pipelined implementation for object detection on embedded systems, identify certain bottlenecks and propose a series of solutions to achieve a better performance. Our proposed detection system can run Fast R-CNN at 1.85fps. Though algorithms evolve rapidly, this work provided an interesting example of system can run Fast R-CNN at 1.85fps. Though algorithms evolve rapidly, this work provided an interesting example of using the same system can run Fast R-CNN at 1.85fps. Though algorithms evolve rapidly, this work provided an interesting example of using the same region proposal extraction algorithms in both training and testing, experiments with bounding box regression achieved a higher mAP. Compared the results in the 2nd row and 4th row of Table 7 with the results shown in the 3rd row and 5th row, though both experiments have used Edge-Boxes and Selective Search for region proposal extraction in test time, training with Edge-Boxes algorithm leads to a 9.0% mAP advantage compared with training with BING.

The impact of bounding box regression is also investigated as the characteristics of the network. Using the same region proposal extraction algorithms in both training and testing, experiments with bounding box regression achieved a higher mAP. Compared the results in the 2nd row and 4th row of Table 7 with the results shown in the 3rd row and the 7th row, on average an mAP gain of 59% relative speedup and the three-stage BING implementation achieves 43% relative speedup. However, we find that three-stage BING implementation greatly increases the power, thus leads to worse energy efficiency compared with the original two-stage pipeline.

### 6.4 Analysis of CNN Model

In this section, we evaluate the influence of different training methods and implementation settings over the accuracy. The results of detection accuracy with different region proposal extraction methods are presented in Table 7. In Table 7, “EB” stands for Edge-Boxes, “SS” stands for Selective Search, and “BBox-reg” stands for bounding box regression.

Previous work of R-CNN [11] has demonstrated the importance of choosing the training dataset. However, in R-CNN, Selective Search is used for both training and testing. We investigate how the different proposal methods in training and testing can be combined and influence the final results. Results from different sets of proposals to train the CNN model for BING-based detector are also presented in Table 7. Training with low-precision proposals deteriorates the mAP by almost a half, probably because true objects are overwhelmed by irrelevant backgrounds. For example, as shown in the 2nd row and the 5th row, though both experiments have used Edge-Boxes and Selective Search for region proposal extraction in test time, training with Edge-Boxes algorithm leads to a 9.0% mAP advantage compared with training with BING.

The impact of bounding box regression is also investigated as the characteristics of the network. Using the same region proposal extraction algorithms in both training and testing, experiments with bounding box regression achieved a higher mAP. Compared the results in the 2nd row and 4th row of Table 7 with the results shown in the 3rd row and the 7th row, on average an mAP gain of 2.15% is led by the bounding box regression.

### 7 CONCLUSION AND FUTURE WORK

In this paper, we propose an energy efficient pipelined implementation for object detection on embedded systems, identify certain bottlenecks and propose a series of solutions to achieve a better performance. Our proposed detection system can run Fast R-CNN at 1.85fps. Though algorithms evolve rapidly, this work provided an interesting example of CPU & GPU cooperation. We also analyzed the bottleneck of the current algorithms on a heterogeneous platform, which could possibly inspire the researchers to develop hardware-friendly algorithms.

As discussed in Section 3.2, most object detection algorithms, like R-CNN and Fast R-CNN, require traditional
bottom-up methods like Selective Search or Edge-Boxes to extract region proposals. The framework is not an end-to-end solution and region proposal methods are currently the speed bottleneck in the framework, especially for our Fast R-CNN implementation with Edge-Boxes. In contrast, the end-to-end object detection method such as Faster R-CNN [26] is able to achieve higher speed compared with Fast R-CNN by nearly cost-free region proposals. We believe the end-to-end model will attract more attention in the future and the search for small but efficient models will never cease. The recent exploration in Neural Network architecture [37] also enables high-accuracy detection with low computational cost.

Finally, we used embedded CPU+GPU platform for the ease-of-development, while FPGA [36] and ASIC [38], [39] had been demonstrated to be more energy efficient than CPU and GPU. A more energy efficient detection solution is expected to realize better low power image recognition systems.

**ACKNOWLEDGMENT**

We would like to thank our FPGA team members Jie Wang, Erjin Zhou, and Jiantao Qi. We also thank Yuzhi Wang for his assistance in programming and debugging before the competition.

**REFERENCES**


Huizi Mao received his B.S. degree in 2016 from Department of Electronic Engineering in Tsinghua University. His research interest is mainly focused on Neural Network acceleration and Software/Hardware co-design.

Song Yao is a Research Assistant in Department of Electronic Engineering, Tsinghua University. He received his B.S. degree in Tsinghua University, China in July 2015. His current research interests include accurate quantization and hardware acceleration for deep learning.

Tianqi Tang received her B.S. degree in 2014 from Department of Electronic Engineering in Tsinghua University, Beijing, China, and she is currently a master student in Tsinghua under the supervision of Prof. Yu Wang & Prof. Rong Luo. Her research mainly focuses on On-Chip Neural Network System, Emerging Non-Volatile-Memory Technology.

Boxun Li received his B.S. degree in Tsinghua University, China in 2013. He is currently a master student in Research Institute of Circuits And Systems, Dept. of E.E, Tsinghua University, supervised by Prof. Yu Wang. His research mainly focused on Energy Efficient Hardware Computing System Design, and Brain Inspired Computing System on Emerging Devices.

Jun Yao received the bachelors and masters degree in computer engineering from Tsinghua University in 2001 and 2004 respectively and he received the doctoral degree in informatics from Kyoto University in 2009. After that he joined the Graduate School of Information Science, Nara Institute of Science and Technology, where in 2013 he became the associate professor in the computer architecture lab. In 2014, he joined the Shannon Cognitive Computing Lab, Huawei Technologies. His recent research interests are cognitive computing and its applications.

Yu Wang (S05-M07-SM14) received the B.S. and Ph.D. (with honour) degrees from the Department of Electronic Engineering, Tsinghua University, Beijing, China, in 2002 and 2007, respectively. He is currently an Associate Professor with the Department of Electronic Engineering, Tsinghua University. His current research interests include parallel circuit analysis, application specific hardware computing (especially on the brain related problems), and power/reliability aware system design methodology. He has authored and co-authored over 130 papers in refereed journals and conferences. He received the IBM X10 Faculty Award in 2010, the Best Paper Award in ISVLSI 2012, the Best Poster Award in HEART 2012, and 6 best paper nominations in ASPDAC, CODES+ISSS, and ISLPED.