cDeepArch: A Compact Deep Neural Network Architecture for Mobile Sensing

Kang Yang[1], Xiaoqing Gong[1], Yang Liu[2], Zhenjiang Li[2], Tianzhang Xing[1]*, Xiaojiang Chen[1], Dingyi Fang[1]
[1]School of Information and technology, Northwest University, China
[2]Department of Computer Science, City University of Hong Kong, Hong Kong

***Abstract*—Mobile sensing is a promising sensing paradigm that utilizes mobile device sensors to collect sensory data about sensing targets and further applies learning techniques to recognize the sensed targets to correct classes or categories. Due to the recent great success of deep learning, an emerging trend is to adopt deep learning in this recognition process, while we find an overlooked yet crucial issue to be solved in this paper — The size of deep learning models should be sufficiently large for reliably classifying various types of recognition targets, while the achieved processing delay may fail to satisfy the stringent latency requirement from applications. If we blindly shrink the deep learning model for acceleration, the performance cannot be guaranteed. To cope with this challenge, this paper presents a compact deep neural network architecture, namely *cDeepArch*. The key idea of the *cDeepArch* design is to decompose the entire recognition task into two lightweight sub-problems: context recognition and the context-oriented target recognitions. This decomposition essentially utilizes the adequate storage to trade for the CPU and memory resource consumptions during execution. In addition, we further formulate the execution latency for decomposed deep learning models and propose a set of enhancement techniques, so that system performance and resource consumption can be quantitatively balanced. We implement a *cDeepArch* prototype system and conduct extensive experiments. The result shows that *cDeepArch* achieves excellent recognition performance and the execution latency is also lightweight.**

I. Introduction

Due to the increasing popularity of mobile devices recently, *e.g.,* smart phones, watches and glasses, together with their equipped rich on-board sensors, like cameras, accelerometers, gyroscopes, wireless modules, etc., *mobile sensing* nowadays emerges as a promising sensing paradigm in a variety of useful application designs [26], [40], [37], [35], [8]. Although their proposed technical details can be dramatically different from one application to another, most of them share a common design principle — utilizing mobile sensors to collect sensory data about the sensing targets and further applying learning techniques to recognize or classify the sensed targets to correct classes or categories to fulfill the application needs [29], [34].

For instance, cameras from smart glasses or smart phones can capture the first-person view of a user [11], so that user's ambient context or the objects captured in the video stream can be inferred and recognized. With such a capability, cognitive aid systems [14], [7] can be developed, where the recognized objects can be used to assist the patients of cognitive decline, e.g., the loss of object and location recognition abilities. The context and object recognition abilities can also enable the Augmented Reality (AR) applications [11], [14], [12]. Following this similar design principle, motion sensors, *e.g.,* accelerometers, gyroscopes and compass, can sense users themselves and then recognize their daily activities in plenty of fitness and e-health designs [21], [5].

*Corresponding author. Email address: xtz@nwu.edu.cn

As the hardware manufacture of on-board sensors is mature, one major factor that dominates above mobile sensing and recognizing performance is the learning techniques adopted to convert the sensory data to the corresponding classes. The traditional machine learning largely relies on the man-crafted features extracted from the data, which highly depends on the quality of the feature selection and the resulting performance is thus known not accurate and reliable enough [29].

The recent great success of deep learning [18] can perfectly avoid such limitations and already demonstrates its strength in many computing fields. Moreover, there are also early attempts made to leverage deep learning to augment the mobile sensing application designs [11], [21], [34]. Although this combination appears as an emerging and promising trend in the community, in this paper, we find the following overlooked yet crucial issue that challenges such an integration and therefore fundamentally limits its end performance:

The amount of recognition targets of deep learning in above applications is normally large, *e.g.,* many daily objects and activities need to be recognized in the cognitive aid [14], [7] and fitness systems [21], [5] respectively, one simple solution to classify such excessive targets is to adopt a very deep neural network [23], [27], but this will naturally lead to huge training and execution latency. More importantly, prevalent processing platforms in our daily life, *e.g.,* the sensing data are offloaded to nearby desktops or laptops for the processing [7], may not fully afford such an overhead to satisfy the stringent latency requirement from applications (in the next section, we will further show that offloading to remote clouds can be also insufficient), *e.g.,* completing within $300ms$ to avoid obvious visual lags [12]. However, if we blindly shrink the deep learning models (*e.g.,* size and depth), the sensing results could become unreliable and inaccurate, *e.g.,* the shrunk model is not able to provide sufficient sensing capabilities to distinguish too many recognition targets. On the other hand, there also lacks a quantitative measure to tell whether the reduced resource consumption really satisfies the available resource conditions for the execution.

To address these challenges above, we propose *cDeepArch* in this paper based on the following insights:

1) If we design a generic neural network model for recognition, the search space is naturally huge and the deep learning model cannot be small, but in *cDeepArch*, we observe that we can decompose the entire recognition task into two sub-problems: *context recognition* and the *context-oriented target recognitions* (*e.g.,* for the objects or activities). As the contexts of a user are normally limited in daily life, we can first design a compact deep learning model for context recognition. Once the context information is identified, the candidate set of the targets to be recognized (associated to this context) will be largely restricted and another compact model will be sufficient for the final target (*e.g.,* objects or activities) recognitions.

2) This decomposition essentially utilizes the storage (adequate on most computing platforms) to trade for the execution efficiency, *e.g.,* breaking one huge and expensive deep learning model into a series of lightweight compact network models under different contexts, so that each compact model merely handles a small subset of objects or activities. To further manage the execution overhead of each compact model to fit for different platforms, we find that the amount of computations of a deep learning model can be mathematically described. By doing this, we can quantitatively "configure" the network model, so that the model size can be maximized according to the resource conditions on the platforms (as its sensing ability generally improves with a larger and deeper model design) and the end performance of recognition can thus be well prompted.

In summary, the contributions of this paper are as follows.

1) We propose a compact deep neural network framework, which decomposes mobile sensing and recognition tasks into two sub-problems, so that compact deep learning models can be comfortably deployed on the resource-lean platforms. In particular, we adopt the first-person video (for the context and object recognitions) and convolutional neural network (CNN) as concrete instances to instrument the design of *cDeepArch*, while the principle can be flexibly extended to other types of sensing data and deep learning models as well.

2) We mathematically formulate the execution latency of the compact neural network model, in terms of floating point computations. The derived formulation can quantitatively guide us to configure the detailed network model size. By doing so, we can obtain a largest model according to the resource conditions from the execution platforms (for maximizing its sensing capability), so as to prompt (together with other our proposed enhancement techniques) the end recognition performance.

3) We develop a prototype system of *cDeepArch* and conduct extensively experiments using public data set Cifar10 and Cifar100 [15]. The results indicate that the execution overhead of *cDeepArch* is lightweight, *e.g.,* the average time to recognize an object or context is about $75ms$ merely. In the meanwhile, it can also achieve very good context and object recognizing performance.

The rest of this paper is organized as follows. We introduce the design preliminary and overview in Sec. II and elaborate the design details of *cDeepArch* in Sec. III. We implement and evaluate *cDeepArch* in Sec. IV. Related works are reviewed in Sec. V before we conclude in Sec. VI.

Resolution (hori. $\times$ vert.)	Video Size (MBps)
3840×2160	6.01
1920×1080	1.99
1280×720	0.98

TABLE I: The measured video sizes under different resolutions with 30 frames per second on iPhone 7 Plus.

II. Design Preliminary and Overview

In this section, we introduce the preliminary and overview of our *cDeepArch* design, including its potential applications, analysis of existing countermeasures to the key design issues, our design goals and the system architecture of *cDeepArch*.

A. Potential applications

Executing deep learning on mobile devices could enable plenty of useful mobile sensing oriented application designs. As aforementioned in Sec. I, in this paper we will focus on the first-person video captured from camera of mobile devices, which can produce at least the following two potential system designs. Of course, if other types of sensing data are adopted [38], [36], more diversified usage can be enabled as well, such as e-health, fitness, etc.

Cognitive aid system: Cognitive decline is a serious disease for the elderly people, which could cause patients the loss of object and location recognition abilities [2]. As such patients may still be able to understand the very basic signal inputs, *e.g.,* simple textual words or sounds, the general solution is to translate the to-be recognized objects, people or locations into their understandable formats to assist them directly. To enable such a design, the camera amounted on mobile devices, *e.g.,* Goolge glass, can capture the first-person view of the patient and the recorded video stream is offloaded for processing by deep learning models for various recognition tasks. Once the recognition is completed, results are sent back to patients in a proper manner, *e.g.,* displaying words or playing a sound.

Augmented reality (AR) system: AR is a live direct view of the physical world through the camera of a mobile device [12]. By intelligently processing the captured first-person video, digital annotations pop out promptly on the screen when the camera is pointed to an object that the user is interested in. For instance, pointing the device to the steak in supermarket receives the tips for cooking and product reviews, and looking at a shopping mall pops up popular restaurants in the building and customers' ratings. In addition, the user can also actively contribute new annotations for updating the AR background database. This tantalizing physical-world viewing capability brings an emerging interface for human-ambiance interactions.

B. Existing countermeasures

Although deep learning is capable to extract ample features from the video frames to conduct precise and reliable recognition tasks for above applications, as stated in Sec. I, the set of

Instance	Processor	vCPU	Memory(GiB)	Price($/h)
c4.large	CPU	2	3.75	0.1
c4.2xlarge	CPU	8	15	0.398
g2.2xlarge	GPU	8	15	0.65

TABLE II: The pricing for three Amazon AWS instances.

the recognition targets is normally large. This naturally incurs a very large and deep network model design, which is however non-trivial to be processed efficiently. To cope with this issue, the widely adopted countermeasures propose to migrate the heavy computations to the remote clouds, *e.g.,* Amazon Web Services [1]. However, we find that such countermeasures suffer at least the following limitations and drawbacks:

1) Long and uncontrollable transmission latency: As the data volume of video frames is relatively large, the latency to transmit video frames to remote clouds could incur unsatisfied delay performance. To unveil this point, we measure the video frame size using an iPhone 7 Plus in Table I. The first column lists three typical video resolutions. For each resolution, the second column is the corresponding video size, which varies from 0.98MBps to 6.01MBps.

If we use the ubiquitous cellular access technique, *e.g.,* 4G and LTE, to transmit video frames, the video resolution to be supported can be highly limited, *e.g.,* the average upload speed of 4G is about 1MBps [3]. More importantly, the service costs will be high as well, which could remarkably prohibit the wide adoption of the upper-layer applications in practice. Other techniques may alleviate this issue, *e.g.,* low-power wireless [13] or Wi-Fi. However, low-power wireless's data rates are usually low [30]. Wi-Fi has sufficient data rates, but recent studies find that its delay performance can be unreliable, *e.g.,* uploading a single frame of size 1Mb can even reach 6 seconds over Wi-Fi [4]. In summary, although clouds are capable to accommodate expensive computations to execute deep learning, the long and unreliable latency to transmit video frames limits its applicability and usability in practice.

2) Service cost: Using remote clouds may also incur an extra service cost. For example, Table II lists the prices of three Amazon AWS instances and their resource parameter values in December of 2017. All of them are sufficient and capable enough to fulfill recognition tasks by CNNs, *e.g.,* c4.large and g2.2xlarge instances can provide second and millisecond level responses, respectively [9]. However, the instance of the lowest price, *e.g.,* c4.large, can even cost over $400 per year with 12 hours usage per day, which is a heavy extra expense.

3) Potential privacy leakage: On the other hand, transmitting user's collected sensory data, *e.g.,* video frames, image and IMU data, to remote cloud servers can suffer potential privacy leakage problems [24], because a mass of user-related data is now no longer physically possessed by the end users but available on cloud, which may expose user's data to attackers and introduce amounts of significant privacy concerns.

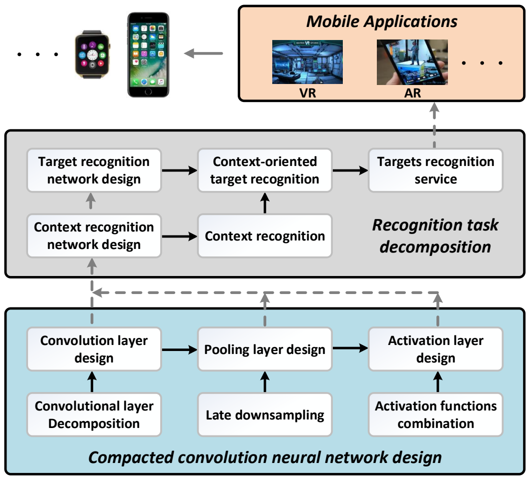
Mobile Applications
VR
AR
Target recognition network design
Context-oriented target recognition
Targets recognition service
Context recognition network design
Context recognition
Recognition task decomposition
Convolution layer design
Pooling layer design
Activation layer design
Convolutional layer Decomposition
Late downsampling
Activation functions combination
Compacted convolution neural network design

Fig. 1: *cDeepArch*'s system architecture.

C. Overview of cDeepArch

With the awareness of above problems, in this paper, we propose to leverage the cloudlet technique [25] (without extra fees) to offload the sensing data to nearby computing platforms and leave remote clouds as the backup when cloudlets are not available, but we do not require any dedicated cloudlet servers in *cDeepArch*. Instead, common desktops or laptops could satisfy the design requirement of *cDeepArch*. As the computing capability rapidly improves, we also envision *cDeepArch* can directly execute on mobile platforms in the near future.

At the high level, the architecture of *cDeepArch* is composed of two primary components: *recognition task decomposition* and *compacted neural network model management* in Fig. 1.

1) Recognition task decomposition: To avoid recognize an object within a huge candidate set using a very large and also deep network model directly, we observe that the presenting of objects is highly coupled with certain context information, *e.g.,* desktops, books and keyboards are often in office or home, and the contexts of a user are normally limited in daily life. Therefore, we propose to decompose the recognition task into two sub-problems in this component: context recognition and context-oriented target recognition, and this component fulfills this decomposition. In particular, this component contains a set of compact neural network models for recognizing user's context information. For each context, it is further associated with another compact neural network model for recognizing the objects that usually appear in such context. The recognized results can be directly used for upper-layer applications.

2) Compact neural network management: The *cDeepArch* design does not assume the availability of dedicated cloudlet servers. Instead, we leverage the available computing resources nearby, such as desktops or laptops. As a result, even we adopt compact neural network models in *cDeepArch*, we still need to carefully control its execution cost within user's preferred or predefined resource budgets on such devices without impact their regular usage. In particular, this component provides a mathematical formulation for the execution overhead of neural

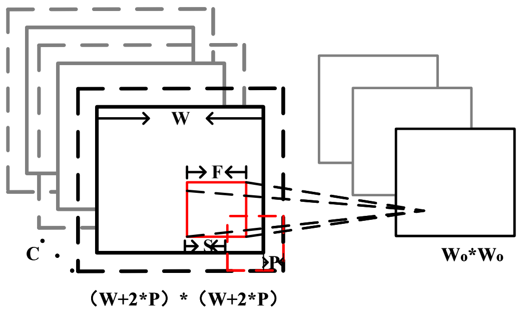

Fig. 2: The convolution operation of one convolutional layer.

network models, based on which, we can precisely manage the model. In addition, this component also contains a set of enhancement modules, *e.g.,* layer separation, late pooling and activation function integration three major techniques, to further augment *cDeepArch*'s recognizing performance (but still satisfying the resource constraint).

III. System Design

We elaborate the design details of compacted neural network model management and recognition task decomposition these two components, following a bottom up order, in this section.

A. Compact neural network model management

For each compact neural network, we need to manage the design of its convolutional layer, pooling layer and activation function for two purposes: 1) the execution of neural network model is within desired resource budget[1] and 2) further prompt its end performance within the resource constraint.

1) Convolutional layer design: The recent studies [11] unveil that the *convolution* operation causes most of computation cost in neural networks, *e.g.,* about 80% [16], and the majority of prior neural network model designs focus on how to configure convolutional layers [27]. Thus, to control the execution overhead of a neural network model, we plan to mathematically formulate the convolutional overhead, denoted as $\mathcal{O}$, so that:

$$\mathcal{O} \quad \leq \quad \alpha \times \mathcal{B}, \tag{1}$$

where $\mathcal{B}$ is the preferred or predefined resource budget and α is the percentage of the computations due to convolution, *e.g.,* we adopt 80% as the default value in *cDeepArch*. In the following, we take the typical neural model structure of three convolutional layers [19] as a concrete example to instrument the formulation, while the principle can be easily extended to other model structures.

Execution overhead formulation. Fig. 2 illustrates the convolution operation of one convolutional layer. During this

[1]As introduced in Fig. 1, *cDeepArch* does not assume the availability of dedicated cloudlet servers, which instead utilizes the available computing resources nearby, such as desktops or laptops. Therefore, even we adopt compact neural network models in *cDeepArch*, we still need to carefully control its execution cost within user's preferred or predefined resource budgets on such devices without impact their regular usage.

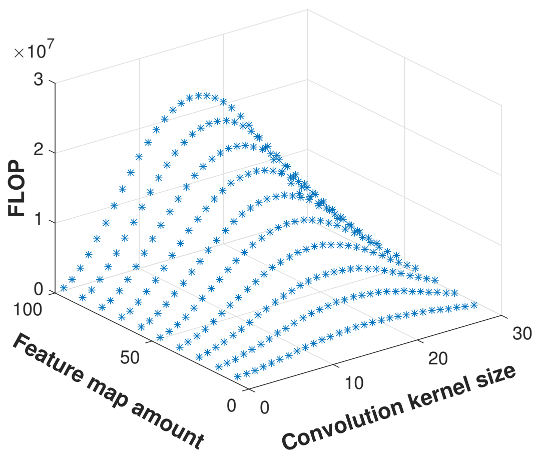

Fig. 3: The FLOP as a function of convolutional kernel size (F) and feature map amount (D), where the input is a 32×32 3-channel RGB frame.

convolution operation, the input data is an original $W \times W$ C-channel picture expanded P (the padding size) pixels, *i.e.,* the new size is $(W+2\times P)\times(W+2\times P)$. The convolution kernel with kernel size F, which moves S (the stride size) pixels each time, is conducted convolution operation with $F \times F$ pixels of the original picture to generate a new pixel on the generated $W_o \times W_o$ feature map.

According to Fig. 2, the execution overhead of a convolutional layer, in terms of FLOP (Floating-point operations), can be formulated as follows. First, output feature map size $W_o \times W_o$ after the convolutional layer can be calculated as:

$$W_o = \frac{W + (2 \times P) - F}{S} + 1, \tag{2}$$

where the input size of this convolutional layer is $W \times W$, and P, F and S represent the padding size, the convolution kernel and the stride size, respectively. Based on Eqn. (2), the overall FLOP of this convolutional layer's execution are:

$$N_{FLOP} = W_o \times W_o \times D \times (F^2 \times C + 1), \tag{3}$$

where D and C stand for the feature map amount and the input channel size respectively, and number 1 is a bias factor. Based on Eqn. (3), the execution overhead of each convolutional layer can be thus calculated through a series of parameters, i.e., W, P, F, S, D, C. Among these 6 parameters, input size W and input channel C can be obtained from the output of the former layer (from the input image size for the first convolutional layer). Meanwhile, the padding size P and stride size S are usually equivalent and small across all convolutional layers in a neural network [28], *e.g.,* padding size and stride size are 2 and 1 respectively. Hence, the execution overhead of each convolutional layer mainly depends on the convolution kernel size F and feature map amount D, which need to be determined by us (as model designer) explicitly.

Without the execution overhead formulation, the neural network model structure is mainly configured based on de-

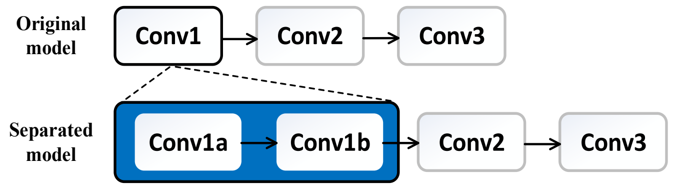

Fig. 4: Convolutional layer separation, where we separate the first convolution layer into two concatenated sub-layers.

signer's experience to balance the execution overhead and the model's accuracy — The designer selects a configuration (based on the past experience or some public models) and then trains the model. If the accuracy after training is low, another configuration will be tested, which lacks a systematic way to configure the model. On the contrary, with the execution overhead formulation, this configuration can be quantitative and effective as follows.

Although a model has multiple convolutional layers, *e.g.,* 3, their sizes are usually not the same, which lead to different execution overhead. Some typical percentages include $(10\%, 60\%, 30\%)$ or $(20\%, 60\%, 20\%)$, e.g., $(10\%, 60\%, 30\%)$ means the execution overhead of the three layers are 10%, 60% and 30%, respectively. Therefore, the execution overhead of layer i, denoted as $\mathcal{O}_i$, should satisfy:

$$\mathcal{O}_i \quad = \quad \mathcal{O} \times \beta_j \leq \alpha \times \mathcal{B} \times \beta_j, \tag{4}$$

where β_j indicates the computation percentage of each layer, $j = 1, 2, 3$ and we adopt the allocation of $(10\%, 60\%, 30\%)$ as the default setting in *cDeepArch*.

On the other hand, based on Eqn. (3), the execution overhead of a convolutional layer can be expressed as the function of convolutional kernel size (F) and feature map amount (D), as illustrated by Fig. 3. Therefore, the constant $\alpha \times \mathcal{B} \times \beta_j$ essentially defines a horizontal plane, which will intersect with the surface in Fig. 3. These intersected positions indicate that the layer's size approaches to the boundary with current F and D values. As the intersected positions may not be unique, we still need to test and compare to finalize D and F, but this method already restricts the feasible configuration within a very limited range, so as to efficiently balance the execution overhead and the end performance, which, to our best knowledge, is never been done before.

Convolutional layer separation. Next we find an additional opportunity to further prompt neural network's recognizing performance yet we still keep the execution overhead within the resource budget. The opportunity is based on a known principle that the recognizing performance generally improves as the depth of the model increases [10]. We can thus separate each layer into two sub-layers to increase model's depth, as in Fig. 4. However, blindly separating one convolutional layer could also increase the execution overhead. We thus need to carefully determine the parameters of each separated sub-layer.

1) Parameter selection. We have formulated the execution overhead of each convolutional layer as N_{FLOP} in Eqn. (3).

The layer separation patterns	Accuracy
1	96%
2	93%
3	93%
1,2	92%
1,3	90%
2,3	87%
1,2,3	20%
Original	89%

TABLE III: The recognition accuracy of all possible convolutional layer separation patterns, compared with the original model for the object recognition.

If we can further similarly formulate for these two sub-layers, denoted as M_{FLOP}, we are able to use the inequality $M_{FLOP} \leq N_{FLOP}$ to conduct the parameter selection. As a matter of fact, there are plenty of feasible ways to configure these two sub-layers. As a practical solution, we set their sizes to be identical and their total execution overhead is thus:

$$M_{FLOP} = 2 \times W_o \times W_o \times D \times (f^2 \times C + 1), \tag{5}$$

where the feature map amount D is the same as the original one and the convolution kernal size f needs to be reconsidered. According to the condition $M_{FLOP} \leq N_{FLOP}$, we have:

$$f \quad \leq \quad \sqrt{(F^2 - 1/C)/2}, \tag{6}$$

which means that we can configure f as $\lfloor \sqrt{(F^2 - 1/C)/2} \rfloor$. Therefore, as long as the kernel size of each sub-layer is set as this value, the layer separation can increase model's depth, and also keeps the execution overhead within the resource budget.

2) Separation pattern. Although model's depth is increased, Eqn. (6), on the other hand, also implies that the kernel size is decreased, which may degrade the recognition performance. To understand this new trade-off, we define 7 different separation patterns in Table III, which contain all possibilities to separate one, two and all three layer(s), respectively.

In this table, we utilize the object recognition accuracy to understand the performance of each separation pattern. The result shows that when all three layers are separated, the accuracy becomes very low, *e.g.,* because all kernels are small in this case. However, when only one or two layer(s) are separated, we indeed observe the improvement. In particular, we find that separating the first layer only achieves the most performance gain. The possible reason is that the first layer has more impacts to extract the primary features from the video frames, and the increased depth could benefit such a feature extraction. Based on the experimental results, we separate the first layer in the current design of *cDeepArch*.

2) Pooling layer design: Another important module of the neural network model is the pooling layer, which is essentially a down-sampling on the extracted features. The purpose of the pooling layer is to reduce the information redundancy and data dimensions of the features to accelerate the training efficiency. However, pooling could also incur the loss of subtle details from features due to its down-sampling nature. In prior

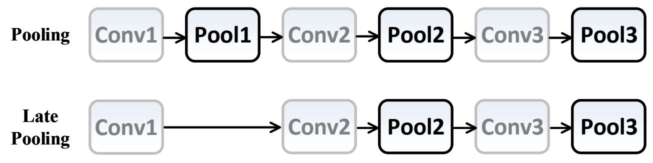

Fig. 5: Illustration to postpone the pooling operation.

model designs, the pooling operation is applied after every convolution layer. In *cDeepArch*, we find that removing the pooling after the first convolutional layer could lead to a better performance, as the down-sampling will not occur for the primary features extracted from the input frames. On the other hand, since the network model itself is compact, there is no obvious impact on the overhead. Our experimental result in Section IV shows that this strategy can effectively improve the recognition accuracy.

3) Activation function design: In addition to the model's structure, *e.g.,* convolutional and pooling layers, the activation function also influences the recognition performance. In the literature, ReLU is the most widely used one as follows [22]:

$$ReLU(x) = \begin{cases} x & x \geq 0, \\ 0 & x < 0, \end{cases} \tag{7}$$

Compared with other activation functions, ReLU is lightweight and thus can remarkably reduce the model training time. However, due to the sparse nature of the ReLU function, most of neurons are suppressed and simply set as zero, which is similar as in our biological nervous system, but this sparsity may limit the learning capability in many situations [31], [20].

To relieve this problem, Batch Normalization (BN) and other ReLU-based activation functions like ELU are proposed in the deep learning field. We can avoid the sparsity drawback, while they usually impose additional computation overhead.

$$ELU(x) = \begin{cases} x & x \geq 0, \\ \gamma \cdot (exp(x) - 1) & x < 0, \end{cases} \tag{8}$$

where γ is scaling factor. In *cDeepArch*, we propose to use both ReLU and ELU two types of functions to complement each other. In particular, we apply ELU to the first convolutional sub-layers and the second convolutional layer to preserve the high quality features extracted from the frames, and ReLU for the last layer to maintain the model to be sufficiently lightweight.

Summary. With compact model management techniques introduced in this section, the designed neural network model can further connect with a SoftMax layer for the classification.

B. Recognition task decomposition

With the compact neural network model management's support from the underlying layer in the protocol stack (Fig. 1), this component is responsible to decompose the entire recognition task into two sub-problems: context recognition and the context-oriented target recognitions (*e.g.,* for the objects or activities). In particular, this component contains a compact neural network model for recognizing user's context information. For each context, it is further associated with another compact neural network for recognizing the objects that usually appear in such a context. The recognized results can be directly used by upper-layer applications, and the configuration of each compact model is managed by the component introduced in the previous subsection.

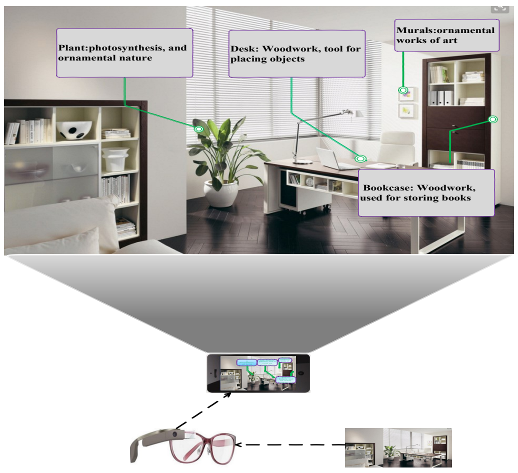
Plant:photosynthesis, and ornamental nature
Desk: Woodwork, tool for placing objects
Murals:ornamental works of art
Bookcase: Woodwork, used for storing books

Fig. 6: Illustration of the cognitive aid system.

For instance, in the cognitive aid system, the users adopt mobile or wearable device to browse the world and the camera from device generates first-person view video frames, so that the objects or humans captured in the video can be recognized and the result is displayed on the screen, as Fig. 6 shows. After the context information, *e.g.,* home, is detected, *cDeepArch* will load another compact recognition model under this context to further conduct object recognitions.

IV. EVALUATION

A. Experimental setup

1) Dataset: We evaluate the performance of our *cDeepArch* design using the public datasets. For the context recognition, we apply the public data set, MIT Place2 [39] that is related to the daily contexts, to train a compact neural network model for the context recognition. For the object recognition, we further adopt Cifar10 and Cifar100 [15], where Cifar10 consists of 32*32 3-channel RGB images in 10 classes, and Cifar100 consists of 32*32 3-channel RGB images with 100 classes. In the evaluation, we divide all classes from Cifar10 and 20 classes from Cifar100 (whose associated contexts are covered by MIT Place2) into different contexts in the daily life, *e.g.,* the classes of airplane, truck and automobile in Cifar10 defined in the airport context. We adopt 400 images per class from Cifar10 and Cifar100 to evaluate the performance, respectively. Prior to the evaluation, we have also collected the datasets for the preprocessing to determine the configuration (e.g., kernel size and feature map) of the compact neural network models. We adopt $1 * 10^8$ FLOP as the resource budget. According to our execution overhead formulation, we limit our configuration of

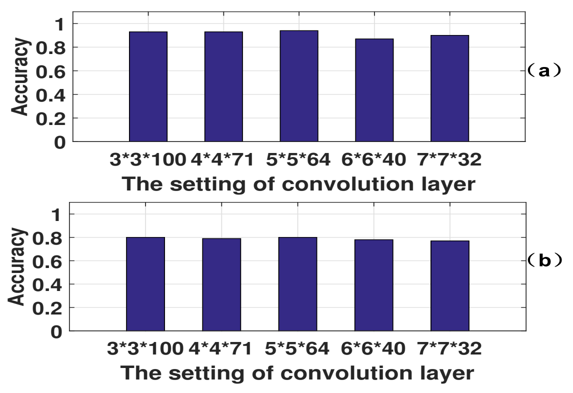

Fig. 7: Convolution kernel size and feature map amount based on the execution overhead formulation for (a) object, (b) context recognition models.

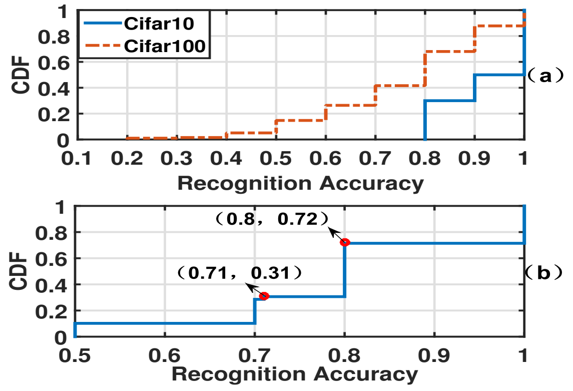

Fig. 9: The recognition accuracy for the (a) object recognition and (b) context recognition performance of *cDeepArch*.

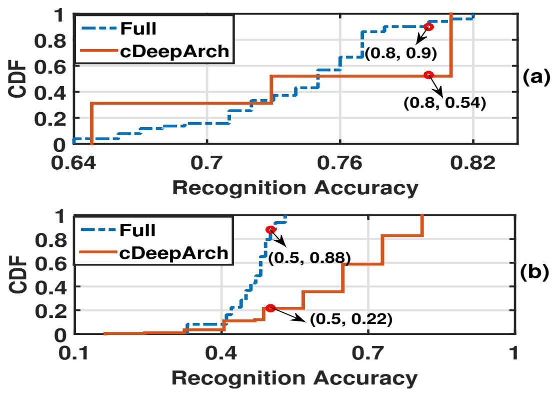

Fig. 8: Overall performance comparison between *cDeepArch* and *Full* on the datesets of (a) Cifar-10 and (b) Cifar-100.

the kernel size and feature map amount to 5 different settings. We develop neural network models as introduced in Sec. III. As shown in Fig.7, the horizontal axis represents the parameter settings, *e.g.,* 3*3*100 means the convolution kernel is 3*3 and there are 100 feature maps. According to the result, we select 5*5*64 as the default setting.

2) Evaluation metrics: We utilize the images from above public datasets to emulate the first-person video application to examine the performance of *cDeepArch*, where we investigate three main performance factors, including the recognition accuracy, parameter amount and time latency. In addition, we also implement an approach, named *Full*, for the comparison, which utilizes a single compact neural network (within the same resource budget) to recognize the objects in all possible contexts, *i.e.,* without the decomposition.

B. Experimental results

Overall performance. Fig. 8 shows the overall performance of *cDeepArch*, where the accuracy is from both the context and object recognition two parts, *i.e.,* the context is recognized correctly first and the object recognition is further correct. For the Cifar10 dataset, Fig. 8(a) shows when the total amount of objects is not large, *e.g.,* 10, both *cDeepArch* and *Full* can achieve similar performance, *e.g.,* their average accuracy is 74.52% and 75% respectively. *Full* sometimes may perform better than *cDeepArch* in this case because it does not involve the errors from the context recognition. However, when the object amount increases, the performance of *Full* rapidly decreases. Fig.8(b) shows that when the object amount is 20, the best performance of *Full* is only around 50%. On the contrary, *cDeepArch* can achieve 81%. In addition, there are about 80% percentages in *cDeepArch* can achieve higher than 50% accuracy and about 20% percentages can reach higher than 80% accuracy. The results directly indicate the effectiveness of the decomposition design in *cDeepArch*.

Object and context recognitions. To further understand the performance of *cDeepArch* achieved in Fig. 8, we now investigate the accuracy of *cDeepArch* for object and context recognitions individually. Fig. 9 shows the *cDeepArch* can achieve promising performance for the object recognition. In particular, for the Cifar10 dataset, the average accuracy is 92% and 70% of cases achieve the accuracy higher than 80%. Similarly, the average and 70% accuracies on the Cifar100 dataset are 78.8% and 60%, respectively.

In Fig. 9, we further plot the accuracy of context recognition achieved by *cDeepArch* over the MIT Place2 data set (§IV-A). The results show that nearly 70% percentages can achieve higher than 70% accuracy and the average accuracy is 81%. However, compared with the object recognition, the accuracy of context recognition is still relatively low, which is the main reason that dominates the overall performance of *cDeepArch* in Fig. 8. In the future, we plan to continue to study how to further improve the accuracy for the context recognition part.

Different enhancement techniques. In Sec. III, we propose the convolutional layer separation, late pooling and activation function fusion three enhancement techniques. In this trial of experiments, we explicitly investigate the effectiveness of each technique. In particular, we only enable one technique each time to train and examine the corresponding models. Then, we use the performance of *cDeepArch* as the baseline and plot the percentage of the performance loss for each technique with respect to the baseline, denoted as D-value in Fig.10, where

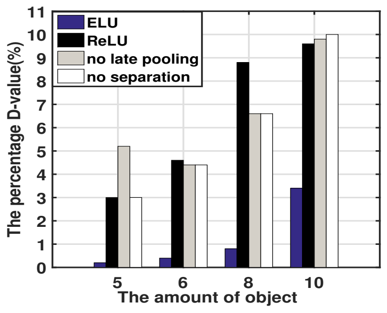

Fig. 10: The object recognition of *cDeepArch* and the performance gain for each enhancement technique on (a) Cifar-10 and (b) Cifar-100.

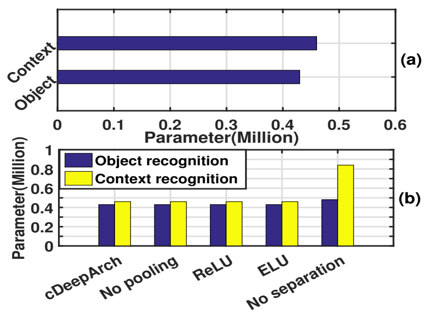

Fig. 11: The parameter amount of (a) *cDeepArch* and (b) different versions.

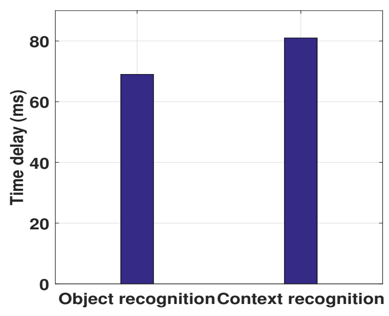

Fig. 12: The time delay for the object recognition and context recognition.

the legends of "ReLU" or "ELU" means all activation layers utilize ReLU or ELU activation function; the "No separation" means the convolutional layer separation technique is not used; the "No late pooling" means the neural network model does not postpone the pooling operation.

In Fig. 10, we further check the performance loss of each technique by varying the amount of objects to be recognized by the compact neural network model. When the object amount is relatively small, *e.g.,* 5, if we do not use these techniques, the performance loss is not significant, *e.g.,* within 5%. This is because in this case the original three layer network model already has enough ability to learn the object features. The enhancement techniques thus contribute to limited improvements. From Fig. 10, we can also observe that when the object amount increases, the performance loss increases as well. For example, the D-value varies from 3% to 10% when the amount of objects becomes to 10, especially for the "ReLU", "No late pooling" and "No separation" three settings. In this case, the network model augmented by the enhancement techniques could demonstrate its advantage compared with the original model structure.

Model parameter comparison. In this experiment, we evaluate the parameter amount of *cDeepArch* which consists of two compacted networks: context recognition network and context-oriented targets recognition network. Fig. 11 shows the parameter amount in million of each network and the impact on the parameter amount under different proposed compacted techniques. From the Fig.11(a), both the context recognition network and the targets recognition network only has small parameter amount, 0.43M or 0.46M. The total parameters of *cDeepArch* is just 0.89M, which indicates *cDeepArch* can only consume few resources. In Fig.11(b), we can further find that using the convolutional layer separation technique can reduce the network parameter obviously. Particularly, in the context recognition network, the parameter amount in million without using separation strategy is almost twice of using it. In addition, the late pooling technique and activation function design also have certain improvements.

The time delay. In this trial of experiment, we investigate the time latency of *cDeepArch* on a common computer with Corei7 CPU. To train each network of *cDeepArch*, it takes 8 to 16 minutes to complete. After the training, as in Fig.12, the time delay of the object recognition is about 69ms and the time delay of context recognition is about 81ms. This indicates that the overall time latency of *cDeepArch* is around 150ms, which can be easily deployed on many existing computing platforms.

V. Related Work

Deep learning for mobile platforms. In the literature, there are some pioneer studies to merge the deep learning with the mobile sensing. In particular, LEO [6] proposes the inference algorithms cross different types of processors and networking resources for the senor inference. MCDNN [9] further introduces an approximation framework for the applications involving continuous vision. Lasagna [21] proposes a hierarchical structure to process the mobile sensing data for understanding and searching. On the other hand, there are also recent works focusing on directly compressing deep learning models. DeepX [17] accelerates the deep learning inference with a software-based design. DeepIoT [33] further proposes a more generic method to compress a set of different neural network model structures.

Given prior achievements, the *cDeepArch* design is parallel to above approaches, *e.g.,* compact neural network models can be further compressed or optimized according to the resource or hardware requirements on different platforms. However, none of these existing studies propose to decompose the original recognition tasks into two lightweight and relevant subproblems, to trade the adequate storage for the CPU and memory efficiency for execution.

Mobile sensing enabled applications. In the literature, there are also plenty of existing application designs enabled by the mobile sensing. For instance, the authors in [7] propose a cognitive assistance system using wearable devices and leverage the cloudlet to process the sensing data. OverLay [12] enables the AR services on user's smart phones. Some other applications are related to the e-health [5], fitness [21], etc. On the other hand, recent works also demonstrate the advantages to apply deep learning into mobile sensing to achieve good system performance. In particular, MobileDeepPill [34] is a smart phone based system that can recognize a large amount of unconstrained pill images. iBlink [32] can further leverage deep learning for facial paralysis patients by smart glasses.

The design of the *cDeepArch* architecture in this paper can potentially benefit above application level designs. When the possible classes or categories to be recognized are relatively large, the deep learning models can be properly decomposed, so that the execution can well match the resource conditions of the processing platforms.

VI. Conclusion

This paper presents *cDeepArch*, a compact neural network model architecture for mobile sensing and recognition. key idea of the *cDeepArch* design is to decompose the entire recognition task into two lightweight sub-problems: context recognition and the context-oriented target recognitions, so that it can be comfortably accommodated by various types of computing platforms. This decomposition essentially utilizes the adequate storage to trade for the CPU resource consumption during execution. To examine the performance of our design, we implement a *cDeepArch* prototype system and conduct extensive experiments. The result shows *cDeepArch* achieves excellent recognition performance and the execution overhead is also lightweight. In the future, we plan to further examine *cDeepArch*'s performance on the mobile platforms.

Acknowledgment

This work is supported by Project NSFC (61602381, 61501372 and 61602382), China Postdoctoral Science Foundation (2017M613187 and 2017M613186), Natural Science Basis Research Plan in Shaanxi Province of China (2017JM6074). This work is also supported by the ECS grant from Research Grants Council of Hong Kong (Project No. CityU 21203516) and the GRF grant from Research Grants Council of Hong Kong (Project No. CityU 11217817).

References

[1] Amazon web services. https://aws.amazon.com.
[2] Definition, causes and symptoms of cognitive decline. https://study.com/academy/lesson/what-is-cognitive-decline-definition-cause-symptoms.html.
[3] How-fast-is-4g. https://www.4g.co.uk/how-fast-is-4g/.
[4] C. Diot and C. Diot. An experimental performance comparison of 3g and wi-fi. In *Proc. of PAM*, 2010.
[5] M. Fritz and B. Schiele. Discovery of activity patterns using topic models. In *Proc. of ACM UbiComp*, 2008.
[6] P. Georgiev, K. K. Rachuri, K. K. Rachuri, and C. Mascolo. Leo: scheduling sensor inference algorithms across heterogeneous mobile processors and network resources. In *Proc. of ACM Mobicom*, 2016.
[7] K. Ha, Z. Chen, W. Hu, W. Richter, P. Pillai, and M. Satyanarayanan. Towards wearable cognitive assistance. In *Proc. of ACM Mobisys*, 2014.
[8] J. Han, C. Qian, X. Wang, D. Ma, J. Zhao, W. Xi, Z. Jiang, and Z. Wang. Twins: Device-free object tracking using passive tags. *IEEE/ACM Transactions on Networking*, 2016.
[9] S. Han, H. Shen, M. Philipose, S. Agarwal, A. Wolman, and A. Krishnamurthy. Mcdnn: An approximation-based execution framework for deep stream processing under resource constraints. In *Proc. of ACM Mobisys*, 2016.
[10] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. In *Proc. of IEEE CVPR*, 2016.
[11] L. N. Huynh, Y. Lee, and R. K. Balan. Deepmon: Mobile gpu-based deep learning framework for continuous vision applications. In *Proc. of ACM Mobisys*, 2017.
[12] P. Jain, J. Manweiler, and R. R. Choudhury. Overlay:practical mobile augmented reality. In *Proc. of ACM Mobisys*, 2015.
[13] X. Ji, Y. He, J. Wang, K. Wu, D. Liu, K. Yi, and Y. Liu. On improving wireless channel utilization: A collision tolerance-based approach. *IEEE Transactions on Mobile Computing*, 2017.
[14] S. J. Kim and A. K. Dey. Simulated augmented reality windshield display as a cognitive mapping aid for elder driver navigation. In *Proc. of ACM SIGCHI*, 2009.
[15] A. Krizhevsky and G. Hinton. *Learning multiple layers of features from tiny images*. Citeseer, 2009.
[16] A. Krizhevsky, I. Sutskever, and G. E. Hinton. Imagenet classification with deep convolutional neural networks. In *Proc. of NIPS*, 2012.
[17] N. D. Lane, S. Bhattacharya, P. Georgiev, C. Forlivesi, L. Jiao, L. Qendro, and F. Kawsar. Deepx: A software accelerator for low-power deep learning inference on mobile devices. In *Proc. of IEEE/ACM IPSN*, 2016.
[18] Y. Lecun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 2015.
[19] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 1998.
[20] B. Liu, M. Wang, H. Foroosh, and M. Tappen. Sparse convolutional neural networks. In *Proc. of IEEE CVPR*, 2015.
[21] C. Liu, L. Zhang, Z. Liu, K. Liu, X. Li, and Y. Liu. Lasagna: towards deep hierarchical understanding and searching over mobile sensing data. In *Proc. of ACM Mobicom*, 2016.
[22] A. L. Maas, A. Y. Hannun, and A. Y. Ng. Rectifier nonlinearities improve neural network acoustic models. In *Proc. of ACM icml*, 2013.
[23] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi. You only look once: Unified, real-time object detection. In *Proc. of IEEE CVPR*, 2016.
[24] K. Ren, C. Wang, and Q. Wang. Security challenges for the public cloud. *IEEE Transactions on Mobile Computing*, 2012.
[25] M. Satyanarayanan, P. Bahl, R. Caceres, and N. Davies. The case for vm-based cloudlets in mobile computing. *IEEE Transactions on Mobile Computing*, 2009.
[26] S. Shen, H. Wang, and R. R. Choudhury. I am a smartwatch and i can track my user's arm. In *Proc. of ACM Mobisys*, 2016.
[27] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *Computer Science*, 2014.
[28] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, et al. Going deeper with convolutions. In *Proc. of IEEE CVPR*, 2015.
[29] J. Wang, Y. Chen, S. Hao, X. Peng, and L. Hu. Deep learning for sensor-based activity recognition: A survey. *Pattern Recognition Letters*, 2017.
[30] J. Wang, S. Lian, W. Dong, X.-Y. Li, and Y. Liu. Every packet counts: Loss and reordering identification and its application in delay measurement. *IEEE/ACM Transactions on Networking*, 2016.
[31] W. Wen, C. Wu, Y. Wang, Y. Chen, and H. Li. Learning structured sparsity in deep neural networks. In *Proc. of NIPS*, 2016.
[32] S. Xiong, S. Zhu, Y. Ji, B. Jiang, X. Tian, X. Zheng, and X. Wang. iblink: Smart glasses for facial paralysis patients. In *Proc. of ACM Mobisys*, 2017.
[33] S. Yao, Y. Zhao, A. Zhang, L. Su, and T. Abdelzaher. Deepiot: Compressing deep neural network structures for sensing systems with a compressor-critic framework. In *Proc. of ACM Sensys*, 2017.
[34] X. Zeng, K. Cao, and M. Zhang. Mobiledeeppill : A small-footprint mobile deep learning system for recognizing unconstrained pill images. In *Proc. of ACM Mobisys*, 2017.
[35] L. Zhang, X.-Y. Li, K. Liu, T. Jung, and Y. Liu. Message in a sealed bottle: Privacy preserving friending in mobile social networks. *IEEE Transactions on Mobile Computing*, 2015.
[36] Y. Zhe, Q. Zhou, L. Lei, Z. Kan, and X. Wei. An iot-cloud based wearable ecg monitoring system for smart healthcare. *Journal of Medical Systems*, 2016.
[37] X. Zheng, J. Wang, L. Shangguan, Z. Zhou, and Y. Liu. Design and implementation of a csi-based ubiquitous smoking detection system. *IEEE/ACM Transactions on Networking*, 2017.
[38] Y. Zheng, G. Shen, L. Li, C. Zhao, M. Li, F. Zhao, Y. Zheng, G. Shen, L. Li, C. Zhao, et al. Travi-navi: Self-deployable indoor navigation system. *IEEE/ACM Transactions on Networking*, 2017.
[39] B. Zhou, A. Lapedriza, J. Xiao, A. Torralba, and A. Oliva. Learning deep features for scene recognition using places database. In *Proc. of NIPS*, 2014.
[40] P. Zhou, Y. Zheng, and M. Li. How long to wait?: predicting bus arrival time with mobile phone based participatory sensing. In *Proc. of ACM MobiSys*, 2012.