Vehicular Communications 10 (2017) 84–90

Contents lists available at ScienceDirect

Vehicular Communications

www.elsevier.com/locate/vehcom

Smart in-car camera system using mobile cloud computing framework for deep learning

Chien-Hung Chen, Che-Rung Lee *, Walter Chen-Hua Lu

National Tsing Hua University, HsinChu, 30013, Taiwan

ARTICLE INFO

Article history:
Received 17 December 2016
Received in revised form 9 July 2017
Accepted 29 October 2017
Available online 7 November 2017

Keywords:
Deep learning
GPU
Mobile cloud

ABSTRACT

Deep learning is becoming a popular technology in various applications, such as image recognition, gaming, information retrieval, for intelligent data processing. However, huge amount of data and complex computations prevent deep learning from being practical on mobile devices. In this paper, we designed a smart in-car camera system that utilizes mobile cloud computing framework for deep learning. The smart in-car camera can detect objects in recorded videos during driving, and can decide which part of videos needs to be stored in cloud platforms to save local storage space. The system puts the training process and model repository in cloud platforms, and the recognition process and data gathering in mobile devices. The mobile side is implemented in NVIDIA Jetson TK1, and the communication is carried out via Git protocol to ensure the success of data transmission in unstable network environments. Experimental results show that detection rate can achieve up to four frame-per-second with Faster R-CNN, and the system can work well even when the network connection is unstable. We also compared the performance of system with and without GPU, and found that GPU still plays a critical role in the recognition side for deep learning.

1. Introduction

Deep learning has become the most promising machine learning approach. From LeCun's LeNet [1] to Khrizevsky AlexNet [2] and recent GoogleNet [3], deep learning has shown its capability of solving difficult computer vision problems [4]. Its detection performance surpasses other artificial classifiers which reply on the hand-crafted features. With the success in vision work, deep learning also attracts attentions from other fields, such as sentiment analysis [5], language processing [6], region of interest localization and description [7], and medical use [8].

The success of deep learning is brought off by three factors: the advance of numerical optimization methods, growth of data volume, and fast computational hardware [9]. New numerical methods solved the convergence problems, which are more and more critical when the number of layers goes deeper and deeper. Large enough training data sets make the extracted features by deep learning sufficiently representative. These required data can be continuously collected by the sensors equipped in modern embedded systems and mobile devices. Last, the accelerators, such as Graphic Processing Units (GPUs), provide strong computing power to support the training of deep learning model.

In the era of Internet of Things (IOT), the deployment of deep models to mobile devices is a nature request. One the one hand, the mobile devices can be smarter with the deep learning ability. Moreover, they also help the data gathering from various sources. On the other hand, the limited power, storage, and computational resources of mobile devices prevent the complex computation, like deep learning, from being practical. Therefore, the mobile cloud computing model, which combines the mobility of mobile devices and the computational resources of cloud platforms via ubiquitously accessible network, becomes a practical solution for mobile applications that utilize deep learning.

Mobile cloud computing has three types of models to coordinate the works between mobile devices and cloud platforms [10]. The first one is to off-load all the work to the cloud platforms, and the mobile devices take care of data input and output only. However, such model usually require frequent data communication, which is not suitable for heavy data transmission. In addition, when the network is unstable, mobile devices cannot work alone. The second model relies on mobile devices to handle most of the works. Such model is only limited to light weighted works that can be processed on mobile devices. The last one splits the works to mobile devices and cloud platforms. This model could balance the

* Corresponding author.
E-mail address: cherung@cs.nthu.edu.tw (W.C.-H. Lu).

https://doi.org/10.1016/j.vehcom.2017.10.001
2214-2096/

computation and communication, but requires good synergy from both sides.

Most of the current solution of mobile deep learning utilizes the first solution since the computation of deep learning is heavy. In this paper, we utilized the third model for deep learning, which puts the training and model repository on the cloud platform, and recognition and data gathering on the mobile devices. Such architecture cuts the storage and computation requirement of mobile devices and the data transmission between cloud and end users. In addition, we employed the Git protocol for data communication data so that the transmissions can be resumed even the network connection is not available sometimes.

We used the smart in-car camera as an example application to demonstrate how the framework works. The smart in-car camera can select suitable deep learning models for video recognition, decide which part of video clips contain important information and send them to cloud platform, and update the deep learning models when necessary. We have implemented the system on NVIDIA Jetson TK1 embedded development board. Experimental results show that detection rate can achieve 2.8 to 4 FPS (frame-per-second) with Faster R-CNN, and the system can work well even when the network connection is unstable. We also compared the performance of system with and without GPU. The result shows about 20 times speedup can be obtained with the acceleration of GPU.

The major contributions of this paper are

- Proposed a split mobile cloud model for deep learning.
- Demonstrate how to use Git protocol in the mobile cloud architecture.
- Implement a smart in-car camera system that can save storage and computation.
- Compare the performance of deep learning inference with and without accelerators.

The rest of this paper is organized as follows. Section 2 introduces background knowledge and related works. Section 3 presents the framework and the implementation details of the smart in-car camera. Section 4 shows the experimental results of system performance. The conclusion and future directions are given in the last section.

2. Background

2.1. Deep learning

Deep learning, or deep neural network, refers to a neural network that consists of one input layer, one output layer and multiple hidden layers. The earliest appearance of deep neural network can be traced to late 1960s in the work published by Alexey Grigorevich and V.G. Lapa [11]. However, deep learning grows at a slow pace due to immature training scheme and architecture in the next few decades. In the 1990s, LeCun trained LeNet, a deep neural net with convolution and pooling layers, with back-propagation algorithm for digit recognition [1]. Stochastic gradient descent was invented in the 1970s to train artificial neural networks.

LeNet is the earliest work to take deep learning into recognition task. In 2006, layer-wise unsupervised pre-training was proposed for parameter initialization [12]. The new training method has two phases. In the pre-training phase, every two consecutive hidden layers are considered a restricted Boltzmann machine and weights are adjusted with an unsupervised learning manner. After pre-training, the whole model is fine-tuned with an additional supervised layer in the second phase. Pre-training makes layer parameters get better values compared to random initialization. As a result, trained models reach more sensible local minimum and quality gets more stable. In 2012, Krizhevsky et al. developed the eight-layer AlexNet and won the ILSVRC 2012 prize with about 85% classification and 66% location accuracy [2]. AlexNet won their competitors who used linear classifier over ten percentage. And the winners of ILSVRC classification task in the following years all used deep neural networks. Instead of using handcrafted classifiers, deep neural network learns high level features during training time. And the ILSVRC challenge results prove its high performance.

With big success in image classification, deep learning attracts focus from other fields. Beside from classification, deep learning is also used in object detection [13–15], speech recognition [16,17], language processing [6], sentiment analysis [5], and many other works. Recent deep learning bloom can be ascribed to new optimization methods, appearance of more powerful processor and rapid growing data volume. With more powerful CPU and multicore processor, especially general purpose GPU, training time can be cut down from months to days. With growing of various data, under-fitting can be prevented and makes deep learning be applied to solve different types of problem. Latest publications not only improve accuracy but boost performance. PReLU activation function pushes classification accuracy over 95% and saves time for deriving gradient [18]. Dropout prevents training from over-fitting [19]. Also, new initialization schemes make pre-training phase unnecessary [20].

The applications of deep learning to object detection for vehicles driving also receives many investigations [21–23]. Unlike them which concerns how to improve the performance of recognition or speed, this paper addresses the issue of system integration for practical considerations, such as package loss and model updates.

2.2. Deep learning packages

2.2.1. OverFeat

OverFeat [15] claims that it is the first work to solve location and classification problem via deep neural network. The team used two networks to handle two vision tasks correspondingly. Regression network and classification network share previous five convolutions with max pooling layers. Then the two networks predict object class and appearance confidence using the same feature map separately. To handle object not at the center of the predicted box or only part of it in the box, input images would be resized to generate different scales of original image. Various scales of images then fed into shared feature extraction layers. Next, max pooling would be conducted on feature maps with different sizes. A fixed sized sliding window goes over all locations of the pooled feature maps and predicts results at each spot. Final result can be determined after merging. OverFeat won ILSVRC 2013 classification and location prize with an under 0.3 error rate.

2.2.2. R-CNN, fast R-CNN and faster R-CNN

Motivated by Alex's success in ILSVRC 12, Girshick et al. developed R-CNN [13] for generalizing convolutional neural network based method to object detection. R-CNN consists of three parts: candidate selection, feature extraction and classification. First, R-CNN chooses selective search to find object proposals in the input images for convenience of comparing with other works. In the feature extraction part, proposals are warped to fix size images and fed into a convolutional neural network. In the last part, a supporting vector machine (SVM) classifies proposals according to feature vectors from previous stage.

Despite R-CNN achieves relatively better result, it is very time consuming during both training and detection time. Fast R-CNN [24] was published to boost performance in 2015. The main bottleneck of R-CNN performance is feeding proposals into deep neural net and calculating features separately. Fast R-CNN introduced ROI pooling layer to share calculated result generated from convolutional neural net. ROI layer extracts fix sized small feature

maps through max pooling on regions in original feature map corresponding to object proposal locations. In this manner, convolutional neural network only needs forward propagation for one time. The other change is Fast R-CNN replaces SVM with a sequence of fully connected layers. This brings different learning process of DNN and SVM to relatively simpler multi-task loss. The changes are rewarded with up to 200× speed up according to [24].

After sharing previous calculation result in Fast R-CNN, selective search becomes performance bottleneck. Therefore, faster R-CNN [25] introduced region proposal network (RPN) to achieve object proposal localization. Final result can reach 5 FPS and 69% mAP on PASCAL VOC 2007 test set on an NVIDIA K40 GPU. Details will be mentioned in the later section.

2.2.3. YOLO

Unlike previous works, YOLO [14] unifies detection into a single stage task. It only uses single network with 24 convolutional layers and two fully connected layers to predict classes and coordinates of possible objects in an image. YOLO divides input image into an SxS grid, by which the deep neural network predicts B boxes and a C dimension class vector for each grid. Only one class would be chosen to represent a grid. As a result, the detection accuracy is hurt while two objects appear in a grid. Another feature of YOLO is a real time detection system. YOLO can predict 45 images in a second and Fast YOLO with lighter model of 9 convolutional layers instead of 24 achieves 155 FPS on PASCAL VOC datasets.

2.3. Mobile cloud computing

In the last decade, mobile devices have been widely used to provide various services, such as gaming, video streaming, healthcare, and position-based services. However, mobile devices are still limited by storage capacity, battery life and computing power. As a result, Mobile Cloud Computing (MCC) came up not long after cloud computing. The Mobile Cloud Computing Forum defines MCC as follows [26,27]:

"Mobile Cloud Computing at its simplest, refers to an infrastructure where both the data storage and the data processing happen outside of the mobile device. Mobile cloud applications move the computing power and data storage away from mobile phones and into the cloud, bringing applications and mobile computing to not just smartphone users but a much broader range of mobile subscribers".

Mobile cloud computing can be considered as a combination of mobile services and cloud computing which communicate through wireless network. The key concept of both cloud computing and MCC is offloading heavy tasks to cloud platforms so that users can access a variety of services without complicate computation on their devices.

The proposed architecture benefits mobile services in many ways. In [27], authors listed three points: extending battery life, improving data storage capacity and processing power, and improving reliability. Battery life time is related to the workload. Processors consume more power if the number of instructions grows or more complicated instructions are executed. MCC offloads heavy workloads to the cloud. As a result, battery life time can be extended. For the second point, IaaS makes users have the authority to access infrastructures which include storage and computing resources. Services are no longer constrained by the hardware limitations of mobile devices. Hence processing power can be improved. By sharing resource on the cloud, MCC also helps utilize servers and saves energy. For the last point, cloud service providers should promise data reliability and security, such as data backup and virus scanning to prevent unexpected data loss or tampered by malicious programs. Executing programs on cloud would be more reliable if data reliability is promised.

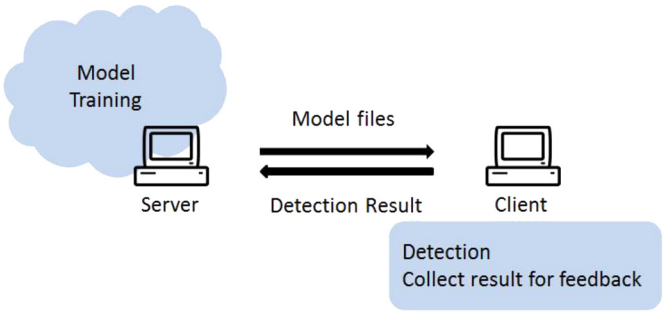

Fig. 1. System architecture.

On the other hand, MCC has issues to overcome. Wireless network is less reliable and has lower bandwidth compared to wired network. Other than bandwidth issue, network for mobile devices is usually unstable, which could result service unavailable sometimes. Discussion of issues includes offloading, security, heterogeneity are remained in [27].

2.4. Git version control system

Git is an open source version control system. The designated work space to be managed is called repository. The system would manage files added to tracked list in the repository. Users can copy files from remote directory through clone command. Cloned repository would be managed by Git system and repository status be kept consistent with remote if git pull command is called. Push command would be used in the situation that a user makes some modification and wants to update file in original repository. Modification of tracked file would not be send back to remote after a commit is created. And according to the official report, Git is faster than many other version control systems.

3. Framework and implementation

3.1. Mobile cloud framework for deep learning

Our design of mobile cloud computing framework for deep learning is based on two facts. The first one is about deep learning processes, and the second one is for mobile network. Deep learning approach has two phases: training and recognition. In the recognition phases, deep learning application only needs to go through forward propagation with input data. After getting output data, some post-processes are required to retrieve desired information. On the contrary, the training phase needs both forward and backward propagation and a large amount of training data. As a result, training process usually takes days or weeks.

For mobile network, according to 4 GTest [28], LTE network has a median of 5.64 Mbps upload bandwidth. Take image process for consideration, common car cameras have 1080p resolution, which also indicates that a raw image can have size overs 6 megabytes. Retrieving results after uploading and processing an image on the cloud becomes infeasible. Even we compress files first, unstable wireless network may prevents users some services.

The proposed architecture is illustrated in Fig. 1, in which the architecture offloads the training phase to the cloud platforms and keeps trained models to the mobile devices for recognition. In addition, the mobile devices collect received data and feed back to cloud. Various data collected from users can help to improve accuracy of deep learning models. Also, the server side can evaluate how the model performs by analyzing prediction result.

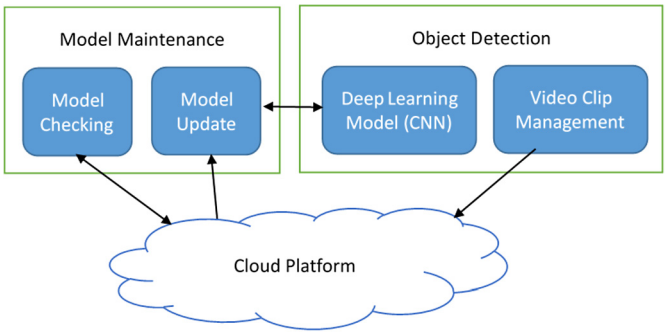

Fig. 2. The system architecture of the smart in-car system for object detection.

3.2. Smart in-car camera system

Many cars have equipped cameras to record videos during driving. However, the capacity of device storage is limited, and many parts of the recorded videos do not have interested objects. With the object detection ability, the car camera can find out the video segments that possibly have objects of interests. Those interested video segments can be uploaded to cloud platforms for further analysis, and the unwanted video segments can be deleted to save storage space. We designed a deep learning object detection system which not only provides detection service but handles model maintenance.

The designed system is presented in Fig. 2. When the system starts, version checker is executed first. If the client does not have model files in local disk storage, it will connect to server and get the desired model. Otherwise the system would check whether an updated version is available. In the second phase, the system has two threads: update thread and detection thread. The update thread keeps checking new model released periodically. This thread is only responsible for model checking and downloading. Models would not be replaced with the updated version until system restarts for safety concern. The detection thread handles data collection. Detection thread would collect network inputs and outputs if possible object appears in the input images. Meanwhile, it keeps a counter to control collected file size. Once the counter hits a designated threshold, the detection thread creates another thread for uploading data. The collected data after upload would be destroyed to save local storage space.

3.3. Detection task

We have surveyed some object detection works, which are introduced in Section 2. We chose Faster R-CNN as our detection approach. Although Faster R-CNN processes is not the best in performance, it has the balanced performance in speed and recognition accuracy.

We have re-produced the C version Faster R-CNN using Caffe [29] and openCV. Fig. 3 introduces the workflow of Faster R-CNN. Although Faster R-CNN can take any size of input, we cropped and resized every input image to a fixes size: 224×224 pixels to make it faster. The following section describes Faster R-CNN implementation in details.

Faster R-CNN trains RPN by sliding a small network over the feature map from the last shared convolutional layer. At each location on the feature map, RPN predicts 3 scales and 3 aspect ratios (the ratio of width to height), which is total 9 proposal regions in the original work. Thus, $3 \times 3 \times W \times H$ region boxes would be predicted for a feature map of size W by H. Centers and sizes of each box is encoded based on a reference box, called anchor box. Let (x_a, y_a) be the center of the anchor box, and (w_a, h_a) be the width and height of the anchor box respectively. Each of box is encoded as a four-tuple (r_w, r_h, s_w, s_h), from which the center of the box (x, y) can be retrieved

$$x = r_w w_a + x_a \text{ and } y = r_h h_a + y_a,$$

and the width w and height h of the box are

$$w = \exp(w_a) s_w \text{ and } h = \exp(h_a) s_h,$$

respectively.

At detection time, region proposal network (RPN) outputs two data blobs, predicted boxes and box scores, after a series of convolutional and pooling processes. Prediction box is represented by four elements (N, C, H, W), where N represents the batch size, C stores 9 four-tuples for proposed boxes, and H and W correspond to the height and the width of feature map. After extracting proposal regions, we drop boxes that are too small and crop the remaining boxes for ensuring that the box size does not exceed input image size. Next, we sort the boxes by confidence c from score blob and apply non-maximal suppression to filter out highly overlapped boxes with lower confidence score.

Classification net takes n filtered regions and feature map generated by the last convolutional layer of region proposal network (RPN) as inputs, and outputs two blobs. Predicted boxes are stored in the same fashion as RPN's output in the first blob. The second blob contains prediction scores of background and desired object classes. If the number of region proposals from RPN exceeds the batch size N, the classification net will run more than one time. The batch size N is adjustable. In Section 4.2, we will discuss how to tune the batch size to optimize performance.

Besides, we also trained the models using py-faster-rcnn [30] with VOC2007 dataset. We trained ZF model with provided definition files and using the alternating optimization [25]. Different from previously described Faster R-CNN, the model takes over intermediate output, which does not require coordinate transformation between RPN and the following layers. Also, the output predicted boxes information are not related with anchor boxes. Instead, it retrieves coordinates of box vertices from the output blob. As to box regression, refined information can be calculated in the same manner with original Faster R-CNN.

3.4. Model maintenance

We chose Git version control protocol for model transmission. There are two reasons of using Git protocol. First, with Git, we save the effort to manage modifications. Second, Git provides the basic authentication service, by which users need account and password to access remote repository.

First time download and follow-up model update can be achieved by Git clone and Git pull functions. And the feedback part can be implemented by Git push. We created two Git repositories on our device: one for storing model files and the other for feedback data. When starts, the system checks whether the model repository exists. If files in need have not been downloaded, Git clone would be triggered to get the latest files from server. Otherwise, the system calls update function to keep version be consistent with the server. In the detection phase, update-thread calls Git fetch constantly and push-thread starts to push feedback repository once collected data reaches a threshold.

Library `libgit2` provides the C APIs and implements Git core functions. Git clone can be triggered simply by calling `git_clone`. As to pull function, `libgit2` achieves the task by calling fetch and merge consecutively. The fetch function only downloads files for repository update. Tracked files would not be changed unless merge function is called. Therefore, we used `git_remote_fetch` in `libgit2` to check the remote status in update thread and `git_merge` is called in version check

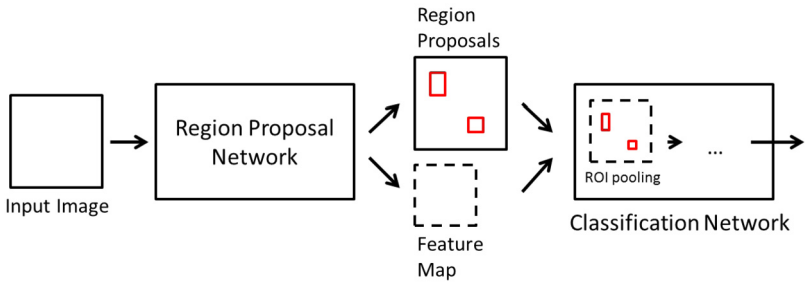

Fig. 3. Faster R-CNN workflow.

phase if model repository already exists. In addition, we used `git_checkout_index` and `git_checkout_tree` to update repository status after merge function called. After the series of merge and update process, changes are committed. Push is implemented similar to merge process; `git_index_add_all` is called to track feedback data before update feedback data repository status. Last, `git_remote_push` is used to finish the task.

In the instable network environment, package loss could cause connection break, which leads to the retransmission of entire models. Since the model file is large, the retransmission takes times to finish, and sometimes, the probability of successful transmission of entire model is impossible if the network is extremely unstable. To avoid this problem, we splits the model files into chunks so that only the unsent chunks need to be retransmitted after the connect is rebuilt. The optimal size of the split chunks depends on the package loss rate, which will be discussed in the experiments.

4. Performance evaluation

We have implemented the smart in-car camera system using NVIDIA Tegra K1, the processor and GPU designed for mobile device. Tegra K1 has four plus one core ARM Cortex-A15 CPU with upto 2.3 GHz frequency and a Kepler GPU. The experimental board has a 2 GB memory, and is connected to wired network. Data can be transmitted through WAN. We chose relatively light weight ZF model provided by origin work to do the detection task. ZF net has 5 convolution layers and 3 fully connected layers. The trained model can be obtained from [25].

We evaluated out work on the trained dataset PASCAL VOC 2007 test data. In addition, we used pedestrian detection database from University of Pennsylvania and the clips downloaded from Youtube, which are videos recorded by car camera on highway.

We have four sets of experiments. The first set of experiments compares the performance with and without GPU. The second one is to tune the batch size for the best performance of the model. The third one compares the system performance for different data sets and different batches. The last one uses simulation to demonstrate how our system works under unstable network environments.

4.1. CPU and GPU performance comparison

We evaluated CPU and GPU performance for recognition process. Fig. 4 shows average CPU and GPU forward pass time. As can be seen, GPU accelerates the forward propagation and performs over 20 times faster than CPU only version, both for RPN and classification net. The significant difference indicates the importance of accelerator. For the rest of our experiments, we only showed the performance results with GPU.

4.2. Batch size tuning

In the recognition phase, the batch size, which is the number of Region Of Interests (ROI) to be processed at the same time,

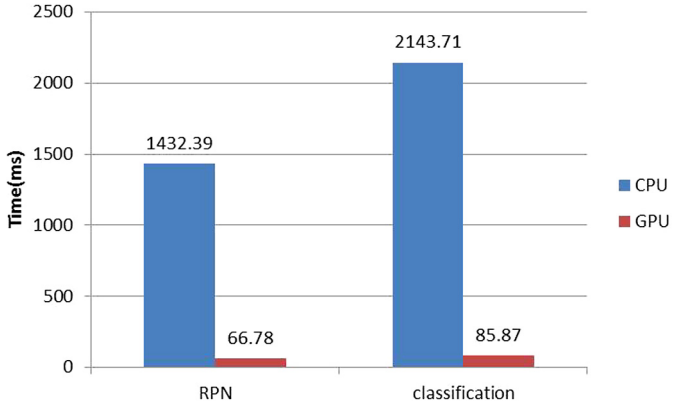

Fig. 4. CPU and GPU performance.

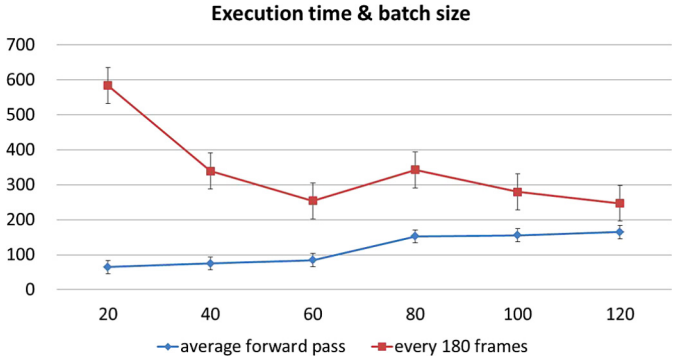

Fig. 5. Performance and batch size.

influences the performance significantly. In this experiment, we evaluated the time (seconds) of forward pass of classification net for different batch size. As Fig. 5 shows, execution time of one iteration grows as the batch size gets larger. This is not a surprised results since larger batch size means more work to do. However, if we normalized the execution time for 180 frames, the time decreases first and then curves up later. This is because when the batch size grows, the number of iterations to process the ROIs of 180 frames decreases, but the time of each iteration increases. The experimental results show the best performance falls around batch size 60, which is the parameter we used in the later experiments.

4.3. System performance

We evaluated the system performance for different number of batches to be processed for the classification net. Since we resized and cropped input images, execution time only relates to the number of candidate Region of Interests (ROI) per image. We can accelerate the system performance by dropping the ROIs with lower scores. The influence of dropping is that some of the objects may not be detected. But experiments show the dropping of

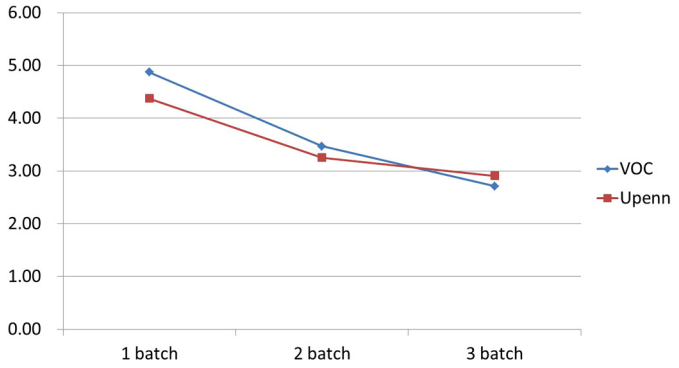

Fig. 6. Performance measured by frames per second.

low score ROIs does not alter the results much (less than 1%). Each frame usually has 60 to 180 ROIs. Since we let batch size be 60, the number of batches per frame is ranged from 1 to 3. Fig. 6 shows the performance result, measured by frame-per-second, for different number of batches. The 1 batch means that we only keep at most 60 ROIs per frame; the 2 batch means each frame can have up to 120 ROIs; and the 3 batch means 180 ROIs are detected for each frame. The results show that the detection task could process over 4 frames per second for 1 batch. When the number of batches grows, the performance drops to around 3 frames per second. The results of two test data have similar behaviors.

It is a clear win of our framework comparing to the result to the framework that places recognition on the cloud. The LTE network has a median of 5.64 Mbps upload bandwidth [28]. For common car cameras, the images usually have 1080 p resolution, which means a raw image takes overs 6 megabytes to store. So the uploading of an image can take over 1 second, and the feedback from cloud takes longer time. So the frame-per-second to process is less than 1. Video compression can significantly reduce the file size to upload, but the compression and decompression will add another level of overhead, which may not achieve better results. Not mention the situation when the wireless network is unstable.

Another metric of performance is the amount of data to upload to the cloud platforms. Comparing to the first type of mobile cloud computing framework, which uploads all the gathered data to cloud, our framework only uploads the frames with interested objects. The number of frames to upload can be reduced to 10% to 1%, depending on where the car is.

4.4. Network transmission

Our smart in-car camera system can dynamically download and update the required models from cloud platform. According to [28], the downlink of current LTE network is much faster than that of the uplink. However, the model files of deep learning usually are usually very large, around hundred megabytes, even with compression. In unstable network environments, transmission of such files could cause a problem, because file transmission needs to restart after re-connection. One solution to reduce the duplicate transmission is to split whole model into chunks with smaller file size. With this approach, duplicate transmitted data size would be limited to the chunk size. On the other side, it needs longer preparation time for file splitting, but that can be done in advance on cloud platform.

Table 1
The transmission time (seconds) of different chunk sizes for different package drop rates.

Drop rate \ No. chunks	1	20	100
0	120	121	130
0.1	133	134	143
0.4	208	202	210
0.5	4382	271	256
0.6	(unable)	2877	451

We used the Markov chain model, proposed in [31], to simulate the package loss to evaluate the performance of the file splitting strategy. The model is presented in Fig. 7, in which the maximum number of retransmission in the simulation is 15, after which the file chunk is required to retransmit. The bandwidth of uplink in simulation is 15 Mbps and the model file size is 224 MB, which is a real data size of ZF model for 20 categories. The latency of each split file transmission is 0.1 second. The performance of three chunk sizes (number of chunks) are compared: 1 (no slitting), 20, and 100. The evaluated package drop rates are 0.1, 0.4. 0.5, and 0.6.

Table 1 summarizes the transmission time of each method in seconds for different package drop rate. The result is the average of 10 times of simulation. It shows when the package drop rate is small, which means the network environment is stable, the splitting method takes longer time owing to the overhead of file transmission latency. However, for larger package drop rate, the time of un-split file grows rapidly. For drop rate 0.6, the un-split file cannot be completed in hours. On the other hand, for chunk number 100, the time grows much slower even when the network is extremely unstable.

5. Conclusion

In this paper, we proposed a mobile cloud computing framework for deep learning. The architecture leaves training on the cloud and performs recognition in the client side with some post processing. The advantage is that the mobile devices can still work under unstable network environment without huge storage and computing capacity. Experiments show that GPU is still critical for the recognition of deep learning, since it can accelerate the computation over 20 times.

Limited storage space and computing power remain a challenge for embedded systems using deep learning. S. Han et al. proposed Deep Compression [32], which reduces storage space by learning only important connections, quantize weights and apply Hoffman encoding. They reduced size of AlexNet from 240 MB to 6.9 MB without accuracy loss. With deeper models applied on mobile devices. For future work, we will integrated their work to the system, and investigate further optimization of storage and computation for deep learning.

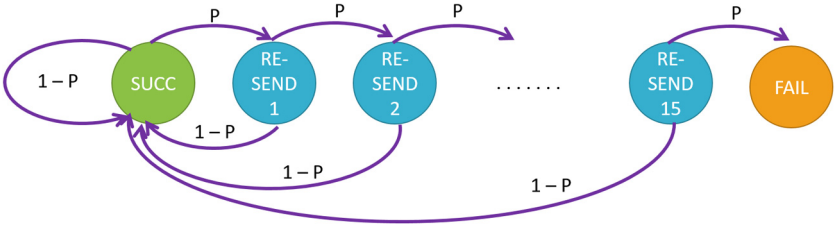

Fig. 7. Markov model for package loss simulation.

References

[1] Y. LeCun, L. Jackel, L. Bottou, A. Brunot, C. Cortes, J. Denker, H. Drucker, I. Guyon, U. Muller, E. Sackinger, et al., Comparison of learning algorithms for handwritten digit recognition, in: International Conference on Artificial Neural Networks, vol. 60, 1995, pp. 53–60.

[2] A. Krizhevsky, I. Sutskever, G.E. Hinton, Imagenet classification with deep convolutional neural networks, in: Advances in Neural Information Processing Systems, 2012, pp. 1097–1105.

[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, A. Rabinovich, Going deeper with convolutions, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2015, pp. 1–9.

[4] Image net challenges, http://www.image-net.org/challenges/LSVRC.

[5] R. Socher, A. Perelygin, J.Y. Wu, J. Chuang, C.D. Manning, A.Y. Ng, C. Potts, Recursive deep models for semantic compositionality over a sentiment treebank, in: Proceedings of the Conference on Empirical Methods in Natural Language Processing, vol. 1631, EMNLP, Citeseer, 2013, p. 1642.

[6] R. Collobert, J. Weston, A unified architecture for natural language processing: deep neural networks with multitask learning, in: Proceedings of the 25th International Conference on Machine Learning, ACM, 2008, pp. 160–167.

[7] J. Johnson, A. Karpathy, L. Fei-Fei, Densecap: fully convolutional localization networks for dense captioning, arXiv preprint arXiv:1511.07571.

[8] R. Fakoor, F. Ladhak, A. Nazi, M. Huber, Using deep learning to enhance cancer diagnosis and classification, in: Proceedings of the International Conference on Machine Learning, 2013.

[9] L. Deng, D. Yu, Deep Learning: Methods and Applications, Tech. Rep., May 2014, https://www.microsoft.com/en-us/research/publication/deep-learning-methods-and-applications/.

[10] S. Abolfazli, Z. Sanaei, E. Ahmed, A. Gani, R. Buyya, Cloud-based augmentation for mobile devices: motivation, taxonomies, and open challenges, CoRR abs/1306.4956, http://arxiv.org/abs/1306.4956.

[11] T. Dettmers, Deep learning in a nutshell, https://devblogs.nvidia.com/parallelforall/deep-learning-nutshell-history-training, 2015.

[12] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, et al., Greedy layer-wise training of deep networks, Adv. Neural Inf. Process. Syst. 19 (2007) 153.

[13] R. Girshick, J. Donahue, T. Darrell, J. Malik, Rich feature hierarchies for accurate object detection and semantic segmentation, in: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2014, pp. 580–587.

[14] J. Redmon, S. Divvala, R. Girshick, A. Farhadi, You only look once: unified, real-time object detection, arXiv preprint arXiv:1506.02640.

[15] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, Y. LeCun, OverFeat: integrated recognition, localization and detection using convolutional networks, arXiv preprint arXiv:1312.6229.

[16] G. Hinton, L. Deng, D. Yu, G.E. Dahl, A.-r. Mohamed, N. Jaitly, A. Senior, V. Vanhoucke, P. Nguyen, T.N. Sainath, et al., Deep neural networks for acoustic modeling in speech recognition: the shared views of four research groups, IEEE Signal Process. Mag. 29 (6) (2012) 82–97.

[17] G.E. Dahl, D. Yu, L. Deng, A. Acero, Context-dependent pre-trained deep neural networks for large-vocabulary speech recognition, IEEE Trans. Audio Speech Lang. Process. 20 (1) (2012) 30–42.

[18] K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: surpassing human-level performance on imagenet classification, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1026–1034.

[19] N. Srivastava, G.E. Hinton, A. Krizhevsky, I. Sutskever, R. Salakhutdinov, Dropout: a simple way to prevent neural networks from overfitting, J. Mach. Learn. Res. 15 (1) (2014) 1929–1958.

[20] I. Sutskever, J. Martens, G.E. Dahl, G.E. Hinton, On the importance of initialization and momentum in deep learning, in: ICML'3, vol. 28, 2013, pp. 1139–1147.

[21] Q. Fan, L. Brown, J. Smith, A closer look at faster R-CNN for vehicle detection, in: 2016 IEEE Intelligent Vehicles Symposium (IV), 2016, pp. 124–129.

[22] S. Lange, F. Ulbrich, D. Goehring, Online vehicle detection using deep neural networks and lidar based preselected image patches, in: 2016 IEEE Intelligent Vehicles Symposium (IV), 2016, pp. 954–959.

[23] J. Hu, T. Xu, J. Zhang, Y. Yang, Fast Vehicle Detection in Satellite Images Using Fully Convolutional Network, Springer, Singapore, 2016, pp. 122–129.

[24] R. Girshick, Fast R-CNN, in: Proceedings of the IEEE International Conference on Computer Vision, 2015, pp. 1440–1448.

[25] S. Ren, K. He, R. Girshick, J. Sun, Faster R-CNN: towards real-time object detection with region proposal networks, in: Advances in Neural Information Processing Systems, 2015, pp. 91–99.

[26] Mobile cloud forum, http://www.mobilecloudcomputingforum.com.

[27] H.T. Dinh, C. Lee, D. Niyato, P. Wang, A survey of mobile cloud computing: architecture, applications, and approaches, Wirel. Commun. Mob. Comput. 13 (18) (2013) 1587–1611.

[28] J. Huang, F. Qian, A. Gerber, Z.M. Mao, S. Sen, O. Spatscheck, A close examination of performance and power characteristics of 4G LTE networks, in: Proceedings of the 10th International Conference on Mobile Systems, Applications, and Services, ACM, 2012, pp. 225–238.

[29] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, T. Darrell, Caffe: convolutional architecture for fast feature embedding, arXiv preprint arXiv:1408.5093.

[30] https://github.com/rbgirshick/py-faster-rcnn.

[31] H.A. Sanneck, G. Carle, Framework model for packet loss metrics based on loss runlengths, in: Proc. SPIE 3969, Multimedia Computing and Networking, 2000.

[32] S. Han, H. Mao, W.J. Dally, Deep compression: compressing deep neural network with pruning, trained quantization and Huffman coding, CoRR arXiv:1510.00149.