공학석사 학위논문

Real-Time Data Offloading for Super Resolution in Mobile Devices

오프로딩을 이용한 모바일 기기에서의 실시간 이미지 초해상도 기술

2018년 8월

서울대학교 대학원

전기·정보공학부

이 주 헌

공학석사 학위논문

Real-Time Data Offloading for Super Resolution in Mobile Devices

오프로딩을 이용한 모바일 기기에서의 실시간 이미지 초해상도 기술

2018년 8월

서울대학교 대학원

전기·정보공학부

이 주 헌

Real-Time Data Offloading for Super Resolution in Mobile Devices

지도 교수 최 성 현

이 논문을 공학석사 학위논문으로 제출함

2018년 7월

서울대학교 대학원

전기·정보공학부

이 주 헌

이주헌의 공학석사 학위 논문을 인준함

2018년 6월

위 원 장:	이 경 무	(인)
부위원장:	최 성 현	(인)
위 원:	윤 성 로	(인)

Abstract

The rapid enhancement of camera performances in smartphones has allowed users to take high quality pictures without high-end digital cameras. However, there still remains a large gap between smartphone cameras and digital cameras when in comes to zoom-in functionality. Most smartphones provide only digial zoom-in functionality, where image quality degradation is inevitable when the user enlarges the image. Even the high-end smartphones embedded with optical lens provide limited optical zoom-in capabilities, leaving users with great inconvenience. While users can employ an external optical lens to utilize the optical zoom-in functionality, having to carry around an extra hardware incurs great overhead, not to mention its price.

Image Super Resolution (SR) can be a solution to overcome this limitation by recovering the quality degradation caused by digital zoom-in. Image SR, a technique to restore high frequency details from a Low Resolution (LR) image to obtain a High Resolution (HR) image, has been a traditional field of research in computer vision. As deep learning based, especially Convolutional Neural Network (CNN) based, methods have shown to outperform traditional methods, and have been actively researched in recent years.

In this paper, we exploit deep learning based image SR to replace the optical zoom-in functionality in smartphones without embedded optical lenses. As there are several resource constraints in smartphones (e.g., computing power, energy, memory), challenges occur when aiming to provide a real-time performance relying solely based on local execution. To tackle the challenge, we propose a server offloading based approach to provide higher frame rate. Through a prototype implementation on Android and extensive experiments in real world environments, we show that our proposed system can provide at least 10 fps.

keywords: Mobile computing, server offloading, deep learning, image super resolution

student number: 2016-28963

Contents

List of Tables

List of Figures

Chapter 1

Introduction

As the performances of cameras in smartphones developed rapidly, users can now take high quality pictures without high-end digital cameras. However, in case of the zoom-in functionality, there is still a large gap between smartphone cameras and digital cameras. As opposed to optical zoom-in, most smartphones provide only digial zoom-in functionality: when the user enlarges certain area of the image to take a closer view, the image is just cropped and resized to fit into the screen, leading to image quality degradation. This limitation of digital zoom-in functionality remains a great annoyance to users.

For users to utilize optical zoom-in functionality, users have two choices: either use high priced models (e.g., Galaxy S9+, iPhone X) embedded with optical lens, or employ additional optical zoom lenses for smartphones. However, even such smartphones embedded with optical lens only provide limited optical zoom in capabilities (x2), and having to carry around additional hardware incurs significant inconvenience to the users, not to mention the additional price for purchase (as high as hundreds of dollars).

Meanwhile, image Super Resolution (SR) can be a solution to recovering the quality of digital zoomed in pictures. Image SR, a technique to restore high frequency details from a Low Resolution (LR) image to obtain a High Resolution (HR) image, has been a traditional field of research in computer vision. However, deep learning

based, especially Convolutional Neural Network (CNN) based methods have recently outperformed traditional methods, and have been actively researched recently.

Although there has been approaches to apply image SR in smartphones, such works focused on restoring the image quality after the pictures are taken [1, 2]. However, such approaches cannot be a fundamental solution, as users are still left with images (degraded by the digital zoom-in) on the preview screen before taking pictures. In this paper, we aim to enable the optical zoom-in functionality in smartphones without optical lenses via deep learning based image SR techniques.

An important issue in designing our system is to determine where the deep learning model inference is executed. As there are several resource constraints in smartphones (e.g., computing power, energy, memory), challenges occur when using the local device for deep learning model execution. While some of recent works on mobile deep learning frameworks for continuous vision applications (e.g., image classification [3, 4] or image segmentation [5]) that rely solely on local execution tackles the challenge by either lowering frame rate (frame per second, or *fps*) or using a low resolution image for input (e.g., 128x128), such compromises are not acceptable in case of image super resolution, as both large input size and real-time performance is required. As an alternative, we propose a server offloading based approach to provide higher frame rate. Through a prototype implementation on Android, we show that about 10 fps can be provided using server offloading.

The major contributions of this paper are as follows:

- We propose an application of applying image super resolution in smartphone cameras. To the best of our knowledge, there has been no literature dealing with the real-time performance of SR on smartphones.
- We implement and train a deep learning based image super resolution model, and verify that it applies well for real pictures taken from smartphone cameras.
- We compare the inference time of the deep learning based image super resolution

model on smartphone and server and verify that server offloading is necessary for providing real-time performance.

- We implement a prototype of our proposed application on Android and evaluate the performance via real world experiment.

The rest of the paper is organized as follows. We summarize preliminary knowledge and related work in Chapter 2. We discuss the overall implementation structure and explain the operational flow in Chapter 3. Performance evaluation of the implemented system is detailed in Chapter 4. We address practical issues of our application in Chapter 5 and conclude the paper in Chapter 6.

Chapter 2

Preliminaries

2.1 Image Super Resolution (SR)

Image Super Resolution (SR) refers to the task of recovering high frequency details from a Low-Resolution (LR) image to generate a High-Resolution (HR) image. Starting from SRCNN [6], deep learning-based approaches have outperformed non-deep learning-based approaches such as A+ [7], and various CNN-based architectures have been proposed [8–10] afterwards to further enhance performance. Current state-of-the-art is EDSR [11], the winner of the NTIRE Challenge 2017 [12].

Several recent work also deal with applying SR in mobile devices. [1] applies non-deep learning based multi-frame SR for pictures taken from smartphone cameras. Google proposed an application for smartphones to reduce internet traffic by downloading an x2 downsampled image and upscaling it in the smartphone via their proposed deep learning based SR model RAISR[1] [2, 14]. Several Android applications such as Camera Super Pixel [15] or TensorZoom [16] also provide similar functionalities. However, all of such approaches aim at applying SR as a post-processing (i.e., after the pictures have been taken); to the best of our knowledge, none of the existing work has dealt with applying SR to preview frames in real time.

[1] RAISR has also been included in Pixel 2 smartphones [13].

2.2 Mobile deep learning framework

As deep learning has proved to be successful in various applications, employing deep learning models on mobile device (e.g., smartphones, tablets)-based applications have drawn a lot of attention in recent years. Existing works on mobile deep learning frameworks can be divided into two branches depending on whether the deep learning model inference is done on the local device or at the server.

2.2.1 Local execution based framework

Local execution based mobile deep learning frameworks do not get help from the others; everything is handled on the device itself. As mobile devices are very short on computational resources, executing deep learning model inference on mobile devices require significant optimization to provide acceptable performance, especially in terms of frame rate or latency. Optimization techniques are applied either at (i) deep learning model level or or at (ii) code level. Deep learning model level optimization involves making the model lightweight (in terms of model size and inference time) via various techniques such as pruning, quantization, and distillation. Code level optimization involves accelerating the computations during the model inference. Popular deep learning frameworks (e.g., Caffe, Torch, and TensorFlow) provide their own mobile versions (e.g. Caffe Android [17], Torch Android [18], and TensorFlow Android [19]), while several other works also propose frameworks to accelerate deep learning model inference [20–23].

Although such optimization techniques are shown to improve the performance significantly, there are some unavoidable limitations: they either provide a low frame rate (e.g., 1 fps) [3, 4], or require a small input image size (e.g., 128x128) [5, 24, 25].

2.2.2 Server offloading based framework

Opposed to local execution based frameworks, server offloading based mobile deep learning frameworks relies on the server for executing deep learning model inference. Server offloading refers to passing the computationally-intensive task on to a dedicated server with abundant resources, i.e., sending the data to be processed to the server and receiving the processed results. Following the works [26] that have exploited server offloading for continuous vision applications where computationally expensive image processing task was offloaded to the server, several approaches propose deep learning applications on mobile devices utilizing server offloading to provide higher frame rate and lower latency [24, 27, 28]. While the biggest merit of server offloading based mobile deep learning framework is that there is no need to worry about the computational cost, meaning there is a high chance where state-of-the-art models can be employed without much optimization, there are some costs that need to be paid: network latency, network cost (e.g., LTE data), and possible server usage cost.

2.3 What is different about SR in smartphones?

2.3.1 Latency

While several recent works focus on the computational efficiency, the primary goal of SR has been enhancing the quality of the generated HR image. As a result, the inference time of the state-of-the-art SR models are often far from real-time performance. For instance, current state-of-the-art model EDSR [11] takes about 100 seconds for x2 upscaling to 2K resolution frame. On the other hand, applying SR in smartphones impose another important challenge: the system should run on real-time, so that users can view the upscaled image from the camera with minimized delay. Large inference time of the SR model can lead to unacceptable latency between the incoming LR frame from the camera and the actual rendering on the smartphone screen, leading to unsatisfactory user experience.

2.3.2 Resource constraints

Furthermore, super resolution for mobile devices incurs challenge as mobile devices are very short on resources (e.g., computing power, memory, and energy). To satisfy the strict latency requirement, deep learning model should be carefully selected and optimized. Even when offloading the data to the server, additional network usage cost and energy consumption should be considered. Furthermore, a naive approach such as applying SR on each incoming frame can lead to a very inefficient system: deciding 'when' and 'how' to apply SR on frames becomes importance for system optimization.

2.4 Local or server?

A question remains when aiming to apply SR in smartphone cameras: where would be perform the deep learning model inference? As mentioned earlier, relying solely on local execution forces us to either gives up high frame rate (i.e., low latency), or large input image size. For some tasks, such sacrifices may be acceptable (e.g., for image classification small input image size may be acceptable). However, we need both the large input image size and low latency for image super resolution, as they both directly affect the user experience. In such a case, relying solely on local execution cannot achieve the goal; the help from a powerful server is a must (we verify this statement by the evaluation in Chapter 4 by showing that even the simplest four counvolutional layered SR model takes about 500 ms in state-of-the-art smartphone). As a result, we focus on server offloading to provide high frame rate and at the same time maintain a large input image size.

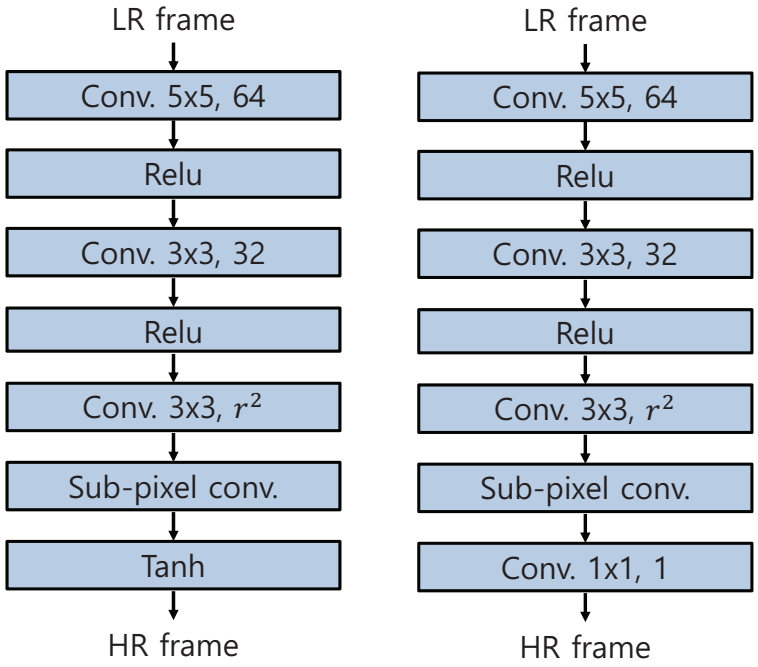
Chapter 3

Implementation

3.1 SR model implementation

We implement ESPCN [8] in TensorFlow [29]. Distinguished from the previous approaches where the input LR frame is upscaled using bicubic interpolation and then passed into the deep learning model, ESPCN takes in the LR frame itself without any upscaling as input. Instead, ESPCN employs a sub-pixel convolutional layer, where the number of channels in the last layer is r^2 times the number of channels in the input LR frame, which are then shuffled together to generate a r times upscaled HR frame. As the input size of the deep learning model is remained the same as the LR frame size, the amount of computation required reduces significantly, and it is reported in the original paper that the inference time of ESPCN for x4 upscaling on 2K Full HD frame is 29 ms on NVIDIA K2 GPU, capable of providing a real time performance.

Figure 3.1(a) depicts the architecture of ESPCN in the original paper. ESPCN is consisted of 3 convolutional layers and a sub-pixel shuffling layer. $Relu$ activation is utilized except for the output layer, where $tanh$ activation is used. Though it is mentioned in the paper that using the $tanh$ activation for the output layer shows a better Peak Signal-to-Noise Ratio (PSNR) performance, we found it difficult to train the model with tanh activation. Instead, we employed a 1x1 convolutional layer for



(a) Original architecture. (b) Modified architecture.

Figure 3.1: The original ESPCN architecture in paper (a) and our modified architecture (b).

scaling the output pixel values, as shown in Figure 3.1(b).

We apply SR on luminance (Y) channel in YCbCr color space, same as the original paper. Aside from the original motivation of this approach that people are more sensitive to the luminance channel, additional benefit arises from having to send only the Y channel values of the frame to the server: this reduces the data rate by three times compared to the case where we apply SR on all RGB channels.

3.2 Prototype implementation on Android

Figure 3.2 depicts the overall prototype architecture. We use Android camera2 API to obtain frames from the camera in the client (smartphone). *CaptureRequest* sets the

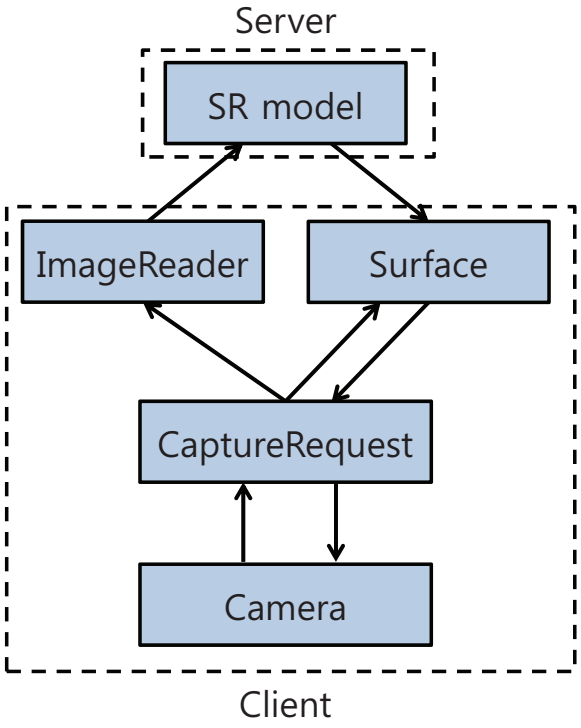


Figure 3.2: Overall implementation architecture on Android.

camera parameters and reads the frames from the hardware. *Surface* displays the camera frame on the screen as preview, reads in user actions (e.g., zoom-in/out, capture), and passes them back to *CaptureRequest* so that the adjusted parameters can be applied. *ImageReader* receives the frames from *CaptureRequest* into a byte array (in YUV format). Afterwards, the Y channel values are extracted and sent to the server, connected by TCP socket. The server, implemented in Python, receives the frame, passes it into the SR model, and sends back the upscaled frame to the client. After the client receives the upscaled frame, YUV pixel values are converted to RGB values [1], copied into a bitmap, which is overlapped to the *Surface*. All the image processing functions, including the YUV to RGB color conversion, are implemented using Java Native Interface (JNI) for better faster runtime performance.

[1]The process of upscaling U,V channel values are included in this process.

Chapter 4

Evaluation

4.1 SR model performance

We trained the ESPCN described in Chapter 3 with DIV2K dataset [12] [1]. Table 4.1 summarizes the performance of the trained model on two benchmark datasets: Set5 and Set14 [2]. We see that ESPCN achieves better performance compared to bicubic interpolation. However, the performance of our current trained model shows about 0.2 dB less performance gain compared to the original paper. We suspect the reason to be the different training dataset, architecture (modifying *tanh* layer to 1x1 convolutional layer, and the specific training parameters not mentioned in the original paper. For our future work, we plan to further train the model with more image datasets as well as fine-tune the training parameters to yield better performance.

We further evaluate the performance of the trained model on actual smartphone pictures. We take 5 pictures from everyday scenes (including indoor and outdoor) using a Galaxy S8, embedded with 4,032x3,024 pixels camera. We take the pictures in raw format (DNG), convert them to PNG format for evaluation. Table 4.2 shows that

[1] The original paper used 50,000 random images from ImageNet [30]. We used different dataset as we could not reproduce the original dataset.

[2] Set5 and Set14 are benchmark datasets, consisted of 5 and 14 images, respectively.

Table 4.1: Mean PSNR (dB) of bicubic interpolation and implemented ESPCN on benchmark datasets

Dataset	Scale	Bicubic interpolation	ESPCN
Set5	x2	30.90	33.58
	x3	28.35	30.85
	x4	26.28	28.48
Set14	x2	27.93	30.14
	x3	25.82	27.67
	x4	24.23	25.86

Table 4.2: Mean PSNR (dB) of bicubic interpolation and implemented ESPCN on smartphone pictures

Dataset	Scale	Bicubic interpolation	ESPCN
Smartphone pictures	x2	34.32	37.37
	x3	31.62	34.29
	x4	29.54	31.82

the trained model also achieves better performance, demonstrating that applying deep learning based SR techniques can be effective.

4.2 Inference time measured on smartphone and server

We compare the inference time of ESPCN on server and smartphone. Server is equipped with NVIDIA GTX 1080 Ti, and we measure the inference time on two smartphones, Galaxy S8 and Nexus 6p, both with Android 7. Table 4.3 shows the core parameters of the compared devices. On Galaxy S8 and Nexus 6p, ESPCN model runs only on CPU using TensorFlow Android [19].

Table 4.4 shows the measured inference time on the above mentioned devices. Input sizes are set to be x8 and x4 downsampled frame size of 12 Mpixel (4,032x3,024

Table 4.3: Specs of server and smartphones.

	GTX 1080 Ti	Galaxy S8	Nexus 6p
Cores	3584	8 (4+4)	8 (4+4)
Clock (GHz)	1.582	2.3/1.7	2/1.5
Memory (GB)	11	4	3

pixels) cameras embedded in several state-of-the-art smartphones, including Galaxy S8 and Nexus 6p[3]. As a general trend, we observe that higher upscale factor does not affect the inference time significantly. This is due to the fact that the upscale factor only affects the number of channels in the sub-convolutional layer. However, as the input size increases, the amount of computation needed increases proportionally throughout each layer, significantly increasing the inference time.

It is important to observe that the inference time on the server and the smartphone differ by more than hundred times. Considering the inference time on mobile devices (as high as few seconds for each frame), relying solely on mobile device for the deep learning model inference cannot satisfy the latency and frame rate needed, as stated in Chapter 2. Note that the tested ESPCN model is consisted of only four convolutional layers; inference time of deeper models are expected to be much larger.

4.3 Latency analysis

In this section, we evaluate the overall latency of our application. The measurement topology is depicted in Figure 4.1. The server uses a and a single NVIDIA GTX 1080 Ti GPU with 11 Gb memory. We use off the shelf TP-link Archer C2600 Wi-Fi Access Point (AP) equipped with QCA9980 chipset and Nexus 6p, equipped with Qualcomm Adreno 430 CPU. The server and the AP are both connected with 1 Gbps LAN, so that

[3]We failed to measure the inference time on Nexus 6p for input size 1,004x756, as the application stopped during the computation.

Table 4.4: Inference time (ms) on server and smartphones.

Size	Scale	GTX 1080 Ti	Galaxy S8	Nexus 6p
504x378	x2	4.0	486.8	1701.1
	x3	4.5	511.8	535.7
	x4	7.0	535.7	2393.6
1008x756	x2	16.0	4010.8	-
	x3	16.0	4516.6	-
	x4	22.5	5243.4	-

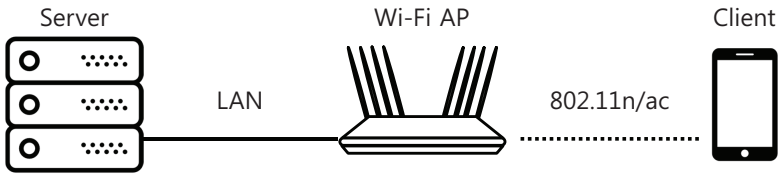


Figure 4.1: Experiment topology.

the wired link does not act as the bottleneck in data transmission.

4.3.1 Offloading latency

We first measure the latency during the server offloading process, which includes the network latency and the inference time of the SR model at the server. Specifically, we measure three latency components: i) *uplink latency*: the time needed for the client to send frame to the server, measured at the server side, ii) inference time of the deep learning model on the server, and iii) *downlink latency*: the time needed for the server to send back the upscaled frame to the client [4].

Impact of wireless link: We first measure the latency for 2 different types of Wi-Fi

[4]Downlink latency is evaluated by measuring the total time between the client sends the frame to the server and receives the upscaled frame at the client side, and subtracting the uplink latency and inference time.

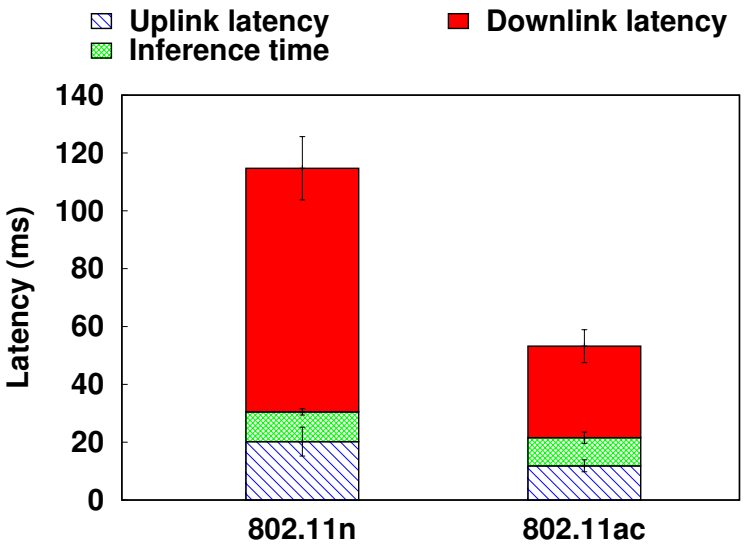


Figure 4.2: Network latency for different Wi-Fi links.

links: 802.11n link in 2.4 GHz band and 802.11ac link in 5 GHz band. The LR image size is 640x360 and the upscale factor is 2. As the uplink and downlink data traffic is in units of Megabytes (about 1.75 and 7 Megabytes respectively), data rate of Wi-Fi affects the latency significantly. Figure 4.2 depicts that using a faster 802.11ac link reduces the *offloading latency* by 54 %. For our subsequent evaluations, measurements were conducted with with 802.11ac link.

Impact of upscale factor: Figure 4.3 depicts the offloading latency with different upscale factors, where the size of the input LR frame is 320x240. Varying the upscale factor for the same input LR frame size increases the downlink data traffic quadratically, significantly increasing the offloading latency. Although the input LR frame size affects the inference time significantly, the overall impact on the offloading latency is small, as the increased inference time is minimal compared to uplink and downlink latency.

Impact of image size: Figure 4.4 compares the offloading latency for varying input LR frame size, where the upscale factor is fixed as 2. Varying the input LR frame size

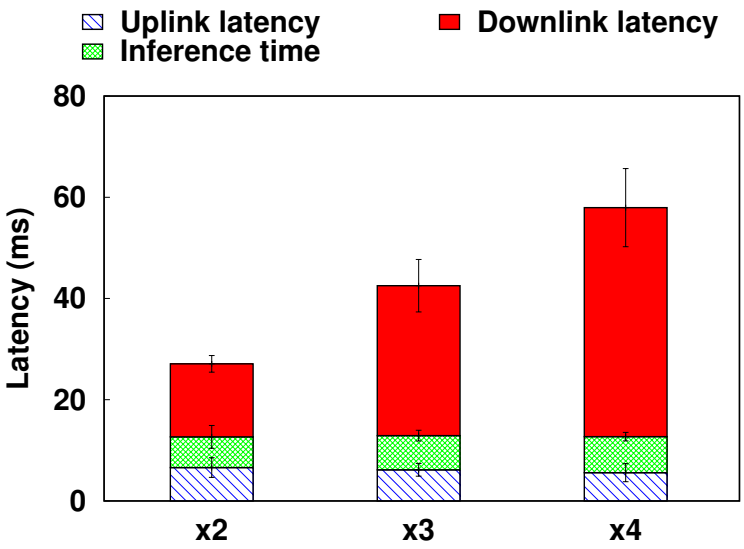


Figure 4.3: Network latency for different upscale factor, where the input LR frame size is 320x240.

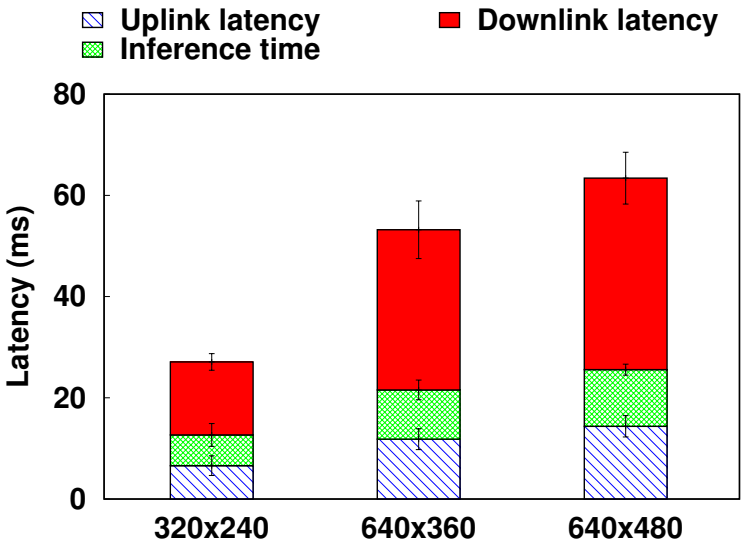


Figure 4.4: Network latency for different input LR frame size, where the upscale factor is 2.

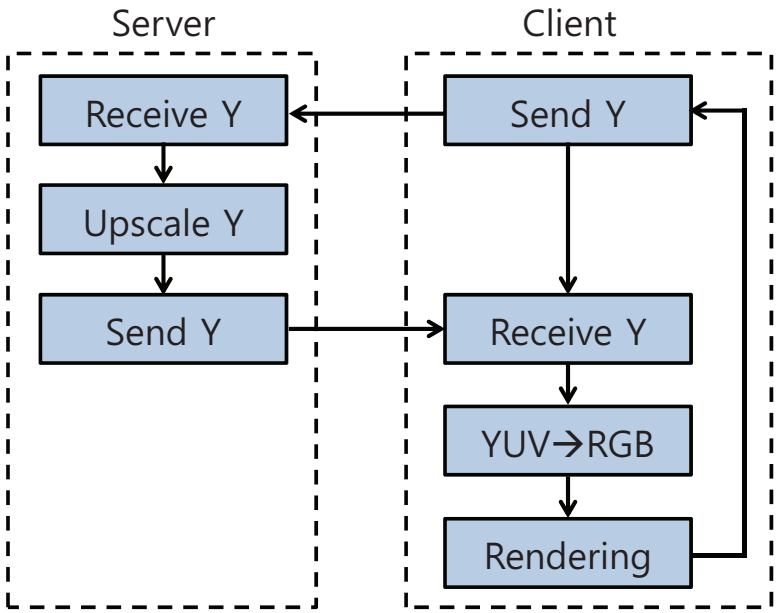


Figure 4.5: Overall operational flow and latency components.

increases both the uplink and downlink data traffic linearly, incurring larger network latency, but not as significantly as the upscale factor. Again, the increased inference time due to higher upscale factor does not affect the overall offloading latency significantly.

4.3.2 Overall latency breakdown

Figure 4.5 depicts the detailed flow of the system. The overall latency can be divided into two blocks: offloading latency and image rendering latency. Image rendering latency includes the conversion from YUV to RGB, setting the converted RGB pixel values in a bitmap, and drawing the bitmap on a canvas for rendering.

Table 4.5 shows the measured latency of each delay components, where the size of the LR frame is 640x360 and the upscale factor is 2. Total latency of the system is 104 ms, resulting in about 10 fps.

In our initial implementation, the image rendering latency is quite substantial, even

Table 4.5: Overall latency breakdown on Galaxy S8 (upscale factor = 2).

Component	Latency (ms)		
	320x240	640x360	640x480
Offloading latency	27	54	63
YUV→RGB conversion	6	18	23
Set RGB pixels in a bitmap	4	9	12
Draw bitmap on screen	1	4	4
Total latency	38	85	93
Frame rate (fps)	26.3	11.8	10.8

comparable to the network latency. This latency can be minimized by using the Android Renderscript API [31] used for running computationally extensive tasks at high performance, which will be included in our future work.

Chapter 5

Discussion

In this section, we discuss some practical issues and challenges related to our proposed application.

5.1 Perceptual quality of generated images

It is a well-known issue that conventional deep learning based SR models that are trained with HR image-downsampled LR image pairs to maximize PSNR (i.e., minimize mean square error) do not always produce images with maximized perceptual quality. The reasons are because (i) , and (ii) various factors other than downsampling contribute to the generation of LR image (i.e., non-ideal downsampling kernel, blurring, noise, artifacts caused by lossy compression, etc.). While [32] derives the theoretical insight on the the tradeoff between the distortion and perception, recent approaches use various training methodologies (e.g., adversarial training with Generative Adversarial Network (GAN)) to generate a HR frame with better perceptual quality [10, 33–35], or propose self-example based approaches where the HR frame generation process is tailored specific to the input LR frame [36, 37].

Though we have seen in Chapter 4 that the trained SR model performs well on actual pictures taken from smartphones in terms of PSNR, the actual effect on the

image quality perceived to the users should be more closely investigated. Note that our proposed system does not require a specific type of SR model to be utilized: any model mentioned above can be employed for in our application as long as the performance and computational efficiency are considered. We plan on testing with various SR models to further enhance the performance of our system.

5.2 Managing high data rate

When using server offloading, optimizing the amount of data traffic between the server and the client is crucial, as they directly affect the energy consumption and cost (e.g., LTE data budget).

State-of-the-art smartphones (e.g., Galaxy S8) are often embedded with 12 Mpixel cameras (4,032x3,024 pixels). Assume a situation where we offload frames maximum zoomed-in (often x8) with frame rate at 10 fps. Even when we send Y channel only so that the data rate is reduced by 3 times compared to sending all RGB channels, the data rate sent to the server becomes,

$$(504 * 378\ pixels) * (1\ channel) * (8\ bits/channel) * (10\ fps) = 15.2\ Mbps. \quad (5.1)$$

One possible alternative may be sending the frames after compression (e.g., in forms of PNG). In such a case, reduced data rate can be estimated as follows,

$$(504 * 378\ pixels) * (1\ channel) * (8\ bits/channel) * (10\ fps) * (0.5^{1}) = 7.6\ Mbps. \quad (5.2)$$

However, such compression can introduce additional latency incurred by the compression process, lowering the overall frame rate.

Note that the above calculation is only for the uplink data traffic (from client to the server). As the size of the upscaled frame is proportional to the square of the upscale factor, downlink data traffic (from server to the client) increases significantly. For instance, when the upscale factor is 2 in the above scenario, the total data rate becomes

$$15.2\ (uplink) + 15.2 * 4\ (downlink) = 76\ Mbps. \quad (5.3)$$

[1]This is the empirical average compression ratio measured on 800 training images in DIV2K dataset

This results in tremendous amount of wireless data usage just by running the application for a few seconds, which may incur huge costs for users especially when LTE link is used.

Mobile deep learning frameworks for continuous vision applications, both including local execution and server offloading, often employ caching mechanism to enhance system efficiency [3, 26]. The core idea of caching mechanism is to reuse the result of previous frames, so that redundant computation can be avoided. As continuous frames from the camera share a lot in common (e.g., background), caching mechanism can effectively reduce the amount of resource usage (e.g., computation, amount of data offloaded to server).

Though caching mechanism has definite merits, it should be employed carefully, as the decision making process to determine whether to reuse previous results or not should not impose much overhead. While some simple methods such as pixel differencing [26] or color histogram matching [3] has been used for image classification or object detection/tracking, caching mechanisms for image super resolution differs from such tasks in two ways: (i) every output pixel value is important and (ii) downlink data traffic from server to client is huge (several times larger than uplink) as opposed to above applications where the result from server is rather simple and small (e.g., label or bounding box of the object). We plan on devising a caching mechanism for image SR in our future work.

Chapter 6

Conclusion

In this paper, we have presented a novel application to enable real-time image super resolution in smartphone cameras. We implemented a deep learning based image super resolution model, and verified that applying SR can be effective on the pictures taken from the actual smartphone cameras. We also checked that the inference time of deep learning models executed on mobile devices are too long to provide real-time performance, and proposed a server offloading based framework to provide lower latency and higher frame rate. Via a prototype implementation on Android, we measured that using server offloading, 10 fps can be provided.

For our future work, we plan to further optimize the latency components in our system. Furthermore, we will devise a caching mechanism adequate for image super resolution to reduce the amount of data offloaded to the server.

Bibliography

[1] N. P. D. Gallego and J. Ilao, "Multiple-image super-resolution on mobile devices: an image warping approach," in *EURASIP Journal on Image and Video Processing*, 2017.

[2] Y. Romano, J. Isidoro, and P. Milanfar, "Raisr: Rapid and accurate image super resolution," in *IEEE Transactions on Computational Imaging*, 2017.

[3] L. N. Huynh, Y. Lee, and R. K. Balan, "Deepmon: Mobile gpu-based deep learning framework for continuous vision applications," in *Proc. ACM MobiSys*, 2017.

[4] X. Zeng, K. Cao, and M. Zhang, "Mobiledeeppill: A small-footprint mobile deep learning system for recognizing unconstrained pill images," in *Proc. ACM MobiSys*, 2017.

[5] B. Zhu, Y. Chen, J. Wang, S. Liu, B. Zhang, and M. Tang, "Fast deep matting for portrait animation on mobile phone," in *Proc. ACM Multimedia Conference*, 2017.

[6] C. Dong, C. Loy, K. He, and X. Tang, "Learning a deep convolutional network for image super-resolution," in *Proc. European Conference on Computer Vision (ECCV)*, 2014.

[7] R. Timotfe, D. Smet, V, and L. V. Gool, "A+: Adjusted anchored neighborhood regression for fast super-resolution," in *ACCV*, 2014.

[8] W. Shi, J. Caballero, F. Huszar, J. Totz, A. P. Aitken, R. Bishop, D. Ruecker, and Z. Wang, "Real-time single image and video super-resolution using an efficient sub-pixel convolutional neural network," in *Proc. IEEE CVPR*, 2016.

[9] J. Kim, J. Lee, and K. Lee, "Accurate image super resolution using very deep convolutional networks," in *Proc. IEEE CVPR*, 2016.

[10] C. Ledig, L. Theis, F. Huszar, J. Caballero, A. Cunningham, A. Acosta, A. Aitken, A. Tejani, J. Totz, Z. Wang, and W. Shi, "Photo-realistic single image super-resolution using a generative adversarial network," in *Proc. IEEE CVPR*, 2017.

[11] B. Lim, S. Son, H. Kim, S. Nah, and K. M. Lee, "Enhanced deep residual networks for single image super-resolution," in *Proc. IEEE CVPR Workshops*, 2017.

[12] R. Timofte, E. Agustsson, L. Van Gool, M.-H. Yang, L. Zhang, and *et al*., "Ntire 2017 challenge on single image super-resolution: Methods and results," in *Proc. IEEE CVPR Workshops*, 2017.

[13] https://www.cnet.com/news/google-pixel-2-photos-get-ai-better-digital-zoom/.

[14] https://ai.googleblog.com/2016/11/enhance-raisr-sharp-images-with-machine.html.

[15] https://play.google.com/store/apps/details?id=com.anforapps.camerasuperpixel&hl=en.

[16] https://play.google.com/store/apps/details?id=uk.tensorzoom&hl=en.

[17] "Caffe android library," https://github.com/sh1r0/caffe-android-lib.

[18] "Torch android," https://github.com/soumith/torch-android.

[19] "Tensorflow android demo," https://github.com/tensorow/tensorow/tree/master/tensorow/examples/android.

[20] S. S. L. Oskouei, H. Golestani, M. Hashemi, and S. Ghiasi, "Cnndroid: Gpu-accelerated execution of trained deep convolutional neural networks on android," in *Proc. ACM Multimedia Conference*, 2016.

[21] N. 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. ACM/IEEE IPSN*, 2016.

[22] M. Alzantot, Y. Wang, Z. Ren, and M. B. Srivastava, "Rstensorflow: Gpu enabled tensorflow for deeplearning on commodity android devices," in *Proc. ACM 1st International Workshop on Embedded and Mobile Deep Learning (EMDL)*, 2017.

[23] Q. Cao, N. Balasubramanian, and A. Balasubramanian, "Mobirnn: Efficient recurrent neural network executionon mobile gpu," in *Proc. ACM 1st International Workshop on Embedded and Mobile Deep Learning (EMDL)*, 2017.

[24] X. Ran, H. Chen, X. Zhu, Z. Liu, and J. Chen, "Deepdecision: A mobile deep learning framework for edge video analytics," in *Proc. IEEE INFOCOM*, 2018.

[25] S. Liu, Y. Lin, Z. Zhou, K. Nan, H. Liu, and J. Du, "On-demand deep model compression for mobile devices: A usage-driven model selection framework," in *Proc. ACM MobiSys*, 2018.

[26] T. Y.-H. Chen, L. Ravindranath, S. Deng, P. Bahl, and H. Balakrishnan, "Glimpse: Continuous, real-time object recognition onmobile devices," in *Proc. ACM SenSys*, 2015.

[27] 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. ACM MobiSys*, 2016.

[28] X. Ran, H. Chen, Z. Liu, and J. Chen, "Delivering deep learning to mobile devices via offloading," in *Proc. ACM VR/AR Network*, 2017.

[29] "Tensorflow," https://tensorflow.org.

[30] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, and *et al.*, "Imagenet large scale visual recognition challenge," in *International Journal of Computer Vision*, 2014.

[31] "Android renderscript api guide." https://developer.android.com/guide/topics/renderscript/compute.html.

[32] Y. Blau and T. Michaeli, "The perception-distortion tradeoff," in *Proc. IEEE CVPR*, 2018.

[33] M. S. Sajjadi, B. Scholkopf, and M. Hirsch, "Enhancenet: Single image super-resolution through automated texture synthesis," in *Proc. IEEE ICCV*, 2017.

[34] A. Ignatov, N. Kobyshev, R. Timotfe, K. Vanhoey, and L. V. Gool, "Dslr-quality photos on mobile devices with deep convolutional networks," in *Proc. IEEE ICCV*, 2017.

[35] X. Wang, K. Yu, C. Dong, and C. C. Loy, "Recovering realistic texture in image super-resolution by deep spatial feature transform," in *Proc. IEEE CVPR*, 2018.

[36] J. B. Huang, A. Singh, and N. Ahuja, "Single image super-resolution from transformed self-exemplars," in *Proc. IEEE CVPR*, 2015.

[37] A. Shocher, N. Coheny, and M. Irani, ""zero-shot" super-resolution using deep internal learning," in *Proc. IEEE CVPR*, 2018.

초 록

최신 스마트폰 카메라의 성능이 빠르게 발전함에 따라, 디지털 카메라 없이도 스마트폰을 이용하여 고화질의 사진들을 손쉽게 찍을 수 있게 되었다. 하지만 줌인 기능의 경우, 스마트폰 카메라와 디지털 카메라 사이에는 여전히 큰 격차가 존재한다. 촬영 영역을 확대해도 화질의 손실이 없는 광학 줌인 기능을 제공해주는 광학 렌즈는 가격상의 문제로 대부분 스마트폰 카메라에 포함되어있지 않으며, 광학렌즈가 포함된 스마트폰의 경우에도 제한된 기능만을 제공한다. 광학렌즈가 부재한 대부분의 스마트폰에서 제공되는 디지털 줌인 기능을 활용하는 경우 촬영할 영역을 확대하면 화질의 손상이 불가피하며, 이는 사용자에게 큰 불편을 준다.

한편, 이미지 초해상도 기술을 이용하는 경우 디지털 줌인된 저화질 사진을 고화질 사진으로 복구할 수 있다. 이미지 초해상도 기술은 저화질 사진으로부터 고주파수 성분들을 추측하여 고화질 사진을 얻어내는 기술로, 전통적인 영상처리 연구 분야 중 하나이다. 최근 딥러닝 연구가 활발해짐에 따라 이미지 초해상도 기술에도 딥러닝이 적용되고 있으며, 합성곱 신경망을 이용한 기술이 기존의 이미지 초해상도 기술들보다 뛰어난 성능을 보임이 확인됨에 따라 딥러닝 기반의 이미지 초해상도 기술이 활발히 연구되고 있다.

본 논문에서는 딥러닝 기반의 이미지 초해상도 기술을 이용하여 디지털 줌인된 사진의 화질을 실시간으로 올려주는 시스템을 제안한다. 스마트폰은 연산처리속도, 에너지 소모, 메모리 등이 제한되어 있기 때문에, 스마트폰 기기 자체에서 딥러닝 모델 연산을 수행하기에 큰 제약이 있다. 본 논문에서는 이러한 한계를 극복하기 위해 서버 오프로딩을 이용하는 시스템을 제안하며, 안드로이드 스마트폰에 프로토타입

구현 후 다양한 환경에서의 실험을 통해 서버 오프로딩을 이용하는 경우 훨씬 높은 주사율을 얻을 수 있음을 보인다.

주요어: 모바일 컴퓨팅, 서버 오프로딩, 딥러닝, 이미지 초해상도 기술
학번: 2016-28963

감사의 글

석사과정을 마무리하며 지난 2년 동안의 연구실 생활을 되돌아보니, 감사할 일들이 너무나 많은 것 같습니다. 항상 제 진로를 먼저 생각하여주시고, 제가 목표를 향해 나아갈 수 있도록 소중한 기회들을 제공해주신 최성현 교수님께 가장 먼저 감사드립니다. 원하는 바를 이루지 못하고 상심해있을 때마다 교수님께서 주신 진심어린 격려와 조언 덕분에 좌절하지 않고 무사히 졸업할 수 있었습니다. 존대말로 학생들을 존중해주시며, 부족한 모습들을 포용하여주시고 뛰어난 통찰력으로 연구를 지도하여주시는 교수님을 보며 진정한 공학자이자 교육자의 모습을 배울 수 있었습니다. 적극적인 자세로 끊임없이 도전하라는 교수님의 말씀 잊지 않고, 앞으로 더욱 발전할 수 있도록 노력하겠습니다.

제 학위 논문 심사에 참여해주신 이경무 교수님, 윤성로 교수님께도 깊이 감사드립니다. 심사위원장으로 참여해주시며 저를 격려해주시고, 지도학생인 임비 연구원님과의 공동연구 기회를 흔쾌히 허락하여주신 이경무 교수님께 감사드립니다. 학부 프로그래밍방법론, 대학원 딥러닝 수업을 통해 제가 코딩과 딥러닝에 관심을 가지게 해주시고, 연구까지 이어나갈 수 있도록 소중한 가르침을 주신 윤성로 교수님께도 다시 한번 감사드립니다. 교수님들께서 주신 소중한 조언들 잊지 않고, 훌륭한 연구자가 될 수 있도록 노력하겠습니다.

제 첫 논문을 지도해주셨던 최재혁 교수님께도 감사의 말씀을 드리고 싶습니다. ‘연구실 선배처럼 생각해달라’고 말씀하시며 많이 부족한 저를 따듯한 격려로 이끌어주신 덕분에 연구자로서 첫 걸음을 성공적으로 딛을 수 있었습니다. 바쁘신 가운데도 제가 지도를 요청드릴 때마다 흔쾌히 시간을 내어주시고 아낌없는 조언을

주신 최재혁 교수님께 다시 한번 감사드립니다.

MWNL의 선후배님들께도 감사의 말씀을 전합니다. 학부 졸업프로젝트 지도조교님으로 인연을 맺고, 연구실 입학 후에도 같은 팀 선배이자 연구자로서 멘토와 같으셨던 구종회 형님께 감사드립니다. 때로는 따듯한 칭찬과 격려로, 때로는 따끔한 충고로 미숙한 제가 더욱 성장할 수 있도록 도와주신 손위평 형님, 날카로운 통찰력으로 제 연구에 조언을 주시던 변성호 형님께도 감사드립니다. 현재 같이 연구를 진행중인, 항상 후배들을 존중으로 대해주시며 선배의 귀감이 되어주시는 김성원 형님께도 감사드립니다. 2년 동안 제가 도움을 요청드릴때마다 항상 마다하지 않고 도와주셨던 이규진, 양창목 형님께도 감사드립니다. 선배지만 저를 동기처럼 대해주시고, 크고 작은 여러 경험을 함께하며 연구실 생활을 더욱 보람차게 만들어주신 황선욱 형님께도 감사드립니다. 긴 시간은 아니었지만 같은 팀에서 함께했던 박승일, 윤호영, 김병준 형님께도 감사드립니다. 이외에도 연구실에서 소중한 시간을 함께 보낸 신연철 형님, 윤강진 형님, 김선도 형님, 손영욱 형님, 김지훈 형님, 이재홍 형님, 김준석 형님, 최준영 형님, 이기택 형님, 곽철영 형님, 박태준 형님, 이지환 형님, 이강현, 허재원, 권휘재, 장민석 형님, 임수훈, 이경진, 이외에도 홈커밍데이, 신년회에서 좋은 말씀들을 아끼지 않으셨던 연구실 선배님들께도 감사를 전합니다.

대학원 생활을 무사히 마칠 수 있도록 도와준 주변의 친구들에게도 감사드립니다. 학부 신입생때부터 지금까지 대학생활을 함께한 동기 김정민, 이호진, 이굳솔, 문지우에게 감사의 인사를 전합니다. 한창 진로고민이 많던 시절 제 고민을 들어주시고, 지금도 항상 저보다 한걸음 먼저 걸어나가시며 뒤따라가는 제게 소중한 조언 아끼지 않으시는 이창현 형님께도 감사드립니다. 고등학교 시절부터 함께 지내온 홍용재, 이영준에게도 감사의 말을 전합니다. 각자의 꿈을 위해 서로 다른 길을 걷고 있지만, 항상 서로의 가장 친한 친구이자 조언자가 되어주는 용재, 영준에게 깊이 감사하며, 앞으로도 소중한 우정 이어나갔으면 좋겠습니다.

사랑하는 여자친구 박수영에게도 감사드립니다. 고민이 있을 때마다 본인의 것처럼 진지하게 들어주고, 진심으로 응원해준 덕분에 항상 든든한 마음으로 연구에 매진할 수 있었습니다. 앞으로 각자의 꿈을 위해 열심히 노력하는 과정에서 의지할 수 있는 남자친구가 될 것을 다짐합니다.

마지막으로, 사랑하는 가족에게 감사드립니다. 지금의 저를 있게 해주신 든든한 인생의 후원자이자 공학도로서의 귀감이 되어주시는 아버지, 항상 본인보다 가족을 먼저 생각하시고 가족들을 위해 배려해주시는 어머니에게 깊이 감사드립니다. 가족의 활력소가 되어주는 동생 주연에게도 고마움을 전하며, 의과대학 본과 공부를 잘 마치고 본인의 꿈을 실현해나가길 기원합니다. 항상 저를 자랑스러워해주시고 따듯하게 맞아주시는 외할아버지, 외할머니, 미국에 계신 친할머니께도 감사드립니다. 앞으로 더욱 열심히 노력하여 자랑스러운 아들, 오빠, 손자가 될 것을 다짐합니다.

MWNL에서의 시간은 제 인생에서 평생 잊지 못할 시간이 될 것 같습니다. 연구실 생활동안 배우고 느낀 부분들 잊지 않고, 앞으로도 더욱 노력하여 MWNL의 자랑스러운 동문이 될 수 있도록 노력하겠습니다.

2018 년 8월

이주헌 올림