High Performance Machine Learning Platform Development and Applications

by

Xu LIU

MANUSCRIPT-BASED THESIS PRESENTED TO ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
IN PARTIAL FULFILLMENT FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY
Ph.D.

MONTREAL, "OCTOBER 14, 2021"

ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

BOARD OF EXAMINERS

THIS THESIS HAS BEEN EVALUATED

BY THE FOLLOWING BOARD OF EXAMINERS

Mr. Abdelouahed Gherbi, Thesis supervisor
Department of Software Engineering and IT, École de technologie supérieure

Mr. Mohamed Cheriet, Thesis Co-Supervisor
Department of Systems Engineering, École de technologie supérieure

Mr. Souheil-Antroin Tahan, Chair, Board of Examiners
Department of Mechanical Engineering, École de technologie supérieure

Mr. Chamseddine Talhi, Member of the Jury
Department of Software Engineering and IT, École de technologie supérieure

Mr. Wubin Li, External Member of the Jury
Ericsson Canada

Mr. Ashkan Ebad, External Independent Examiner
National Research Council Canada

THIS THESIS WAS PRESENTED AND DEFENDED

IN THE PRESENCE OF A BOARD OF EXAMINERS AND THE PUBLIC

ON "NOVEMBER 24, 2021"

AT ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

ACKNOWLEDGEMENTS

Firstly, I would like to thank my supervisor, Professor Abdelouahed Gherbi, and co-supervisor, Professor Mohamed Cheriet, for their professional guidance, unlimited support, and precious suggestions during my Ph.D. study.

I would also like to thank my former co-supervisor, Wubin Li, for his full help, cherishable opinion in my learning and daily life.

Also, I would convey my most profound appreciation to my family, especially my parents, Shiqi Liu and Aixia Qi, for their unconditional love, support, and great patience throughout my whole life.

Also, I would like to thank Ericsson Canada for giving me the precious intern opportunity, which gave me a meaningful industrial experience.

Finally, I would like to thank my co-authors and other friends for their supports and help during the Ph.D. journey.

Plate-forme d'apprentissage automatique haute performance : développement et applications

Xu LIU

RÉSUMÉ

L'apprentissage automatique (ML) et l'apprentissage profond (DL) sont en plein essor. Cependant, les algorithmes sous les ML et les DL impliquent beaucoup de travail informatique à haute densité. L'unité centrale de traitement (CPU) traditionnelle n'a que peu de threads de calcul. Et chaque thread doit exécuter des instructions en séquence. Ainsi, même les processeurs possèdent une fréquence plus élevée, ils doivent traiter des dizaines de milliers de tâches informatiques une par une. Au contraire, Graphics Processing Unit (GPU) et Field Programmable Gate Array (FPGA) possèdent souvent des dizaines de milliers d'unités de calcul, qui peuvent efficacement effectuer des calculs parallèles pour améliorer considérablement les performances d'entraînement et d'inférence du modèle.

Nous proposons dans cette thèse une nouvelle méthodologie de conception hybride GPU-FPGA pour relever le défi ci-dessus de formation et d'inférence de calcul haute densité. Selon la méthodologie de conception, nous avons développé une nouvelle plate-forme ML ou DL hétérogène basée sur GPU-FPGA. Étant donné que les algorithmes d'apprentissage sont souvent modifiés et que la programmation GPU est beaucoup plus simple et flexible que les FPGA, la phase d'apprentissage est implémentée sur le GPU. Sinon, nous effectuons la phase d'inférence sur le FPGA en nous basant sur les deux raisons suivantes : La première est que nous modifions à peine la conception de l'algorithme d'inférence. L'autre est que les FPGA ont une efficacité énergétique plus élevée et un délai inférieur à celui des GPU. De plus, étant donné que les deux plates-formes ont des formats de fichier de modèle différents et ne peuvent pas être substitués directement, nous avons conçu un convertisseur de modèle entre les deux phases pour convertir le modèle de la plate-forme d'entraînement à la plate-forme d'inférence.

Pour évaluer la méthodologie ci-dessus et les performances de la plate-forme, nous avons mis en place un réseau neuronal convolutif (CNN) pour reconnaître les chiffres manuscrits et un réseau neuronal profond (DNN) pour prédire l'efficacité d'utilisation de l'énergie (PUE) du centre de données avec la méthodologie de conception hybride sur l'hétérogénéité Plate-forme. De plus, les résultats expérimentaux ont montré que notre approche a obtenu une amélioration significative des performances sur la formation et l'inférence ML.

De plus, afin d'évaluer pleinement notre conception ML hybride, nous avons étendu les expériences pour inclure deux approches de reconstruction spectrale. Ces deux méthodes sont conçues pour résoudre le problème gravement sous-contraint de la reconstruction d'images multispectrales à partir d'images RVB. Le cœur de ces deux méthodes tourne autour de la façon de générer des informations multispectrales qui sont perdues en raison de la compression.

La première approche est basée sur Variational Autoencoder (VAE) et Generative Adversarial Network (GAN). Le VAE extrait les informations sur les caractéristiques clés des images RVB

d'entrée via l'encodeur, les reparamétre avec une valeur échantillonnée au hasard à partir de la distribution normale, puis restaure les sorties de type MSI via le décodeur. Le GAN est chargé d'entraîner le générateur à générer des images de type MSI à partir de vecteurs latents reparamétrés. Le GAN est chargé d'apprendre au générateur à créer des images de type MSI à partir de vecteurs latents reparamétrés. Le problème de la reconstruction des MSI à partir de RVB peut être résolu avec un faible coût de calcul.

La deuxième méthode est appelée Taiji Generative Neural Network (TaijiGNN), qui combine le GAN cyclé et l'ancienne philosophie chinoise "Taiji". TaijiGNN se compose d'une paire de générateurs fonctionnant dans des directions opposées. La sortie d'un générateur est connectée à l'entrée de l'autre, formant une structure en boucle. Cette structure de boucle peut faire passer l'entrée à travers le domaine de sortie, puis revenir au domaine d'entrée. Par conséquent, TaijiGNN peut convertir le problème de la comparaison d'images dans différents domaines en un problème de comparaison d'images dans le même domaine. Dans le même domaine, le problème sévèrement sous-contraint évoqué ci-dessus peut être résolu naturellement. De plus, TaijiGNN a absorbé l'essence du Taiji pour entraîner la paire de générateurs. Pendant le processus d'entraînement, la paire de générateurs fonctionne comme un couple, utilisant leurs propres avantages pour se compléter, s'aidant mutuellement à atteindre la convergence, de sorte que l'ensemble du système entre dans un état d'équilibre dynamique, très similaire au "Yin" de Taiji. " et " Yang " mode de travail bipolaire.

Nous utilisons deux ensembles de données spectrales classiques, CAVE et ICVL, pour évaluer VAE-GAN et TaijiGNN. Et ces deux approches utilisent beaucoup moins de données d'entraînement que l'état de l'art et atteignent ou dépassent leurs résultats. De plus, les deux approches sont implémentées et vérifiées sur une plate-forme informatique hétérogène conçue à l'aide de notre méthodologie hybride d'apprentissage automatique. Leurs vitesses d'entraînement et d'inférence ont été grandement améliorées.

Mots-clés: apprentissage automatique, l'apprentissage en profondeur, calcul GPU, informatique FPGA, calcul haute performance, réseau accusatoire génératif, autoencodeurs variationnels, image multispectrale, reconstruction spectrale et traduction, traitement d'image, VAE-GAN, Taiji GNN.

High Performance Machine Learning Platform Development and Applications

Xu LIU

ABSTRACT

Machine learning (ML) and Deep learning (DL) are booming. However, the algorithms beneath the MLs and the DLs involve much high-density computing work. Traditional Central Processing Unit (CPU) only has few computing threads. And each thread must execute instructions in sequence. So even the CPUs own a higher frequency, they have to process tens of thousands of computing jobs one by one. On the contrary, Graphics Processing Unit (GPU) and Field Programmable Gate Array (FPGA) often own tens of thousands of computing units, which can efficiently conduct parallel computing to enhance the model's training and inferencing performance dramatically.

We propose in this thesis a novel hybrid GPU-FPGA-based design methodology to address the above training and inferencing high-density computing challenge. According to the design methodology, we developed a new heterogeneous GPU-FPGA-based ML or DL platform. Since the training algorithms are often changed, and the GPU programming is much easier and more flexible than FPGAs, the training phase is implemented on the GPU. Otherwise, we perform the inferencing phase on the FPGA based on the following two reasons: One is that we scarcely change the inferencing algorithm design. The other is FPGAs have higher energy efficiency and lower delay than GPUs'. Moreover, since the two platforms have different model file formats and can not be substituted directly, we designed a model converter between the two phases to convert the model from the training platform to the inferencing platform.

To evaluate the above methodology and platform's performance, we have implemented a convolutional neural network (CNN) for recognizing handwritten digits and a deep neural network (DNN) for predicting the data center's Power Usage Effectiveness (PUE) with the hybrid design methodology on the heterogeneous platform. Moreover, the experimental results presented that our approach has gained significant performance improvement on the ML training and inferencing.

In addition, in order to fully evaluate our hybrid ML design, we extended the experiments to include two spectral reconstruction approaches. Both of these methods are designed to solve the severely under-constrained problem of reconstructing multispectral images from RGB images. The core of these two methods revolves around how to generate multispectral information that is lost due to compression.

The first approach is based on Variational Autoencoder (VAE) and Generative Adversarial Network (GAN). The VAE extracts the key feature information from the input RGB images through the encoder, reparameterizes it with a value randomly sampled from the normal distribution, and then restores the MSI-like outputs through the decoder. GAN is responsible for training the generator to generate MSI-like pictures from re-parameterized latent vectors. GAN is accountable for teaching the generator to create MSI-like images from re-parameterized latent

vectors. The problem of reconstructing MSIs from RGB can be solved with low computational cost.

The second method is called Taiji Generative Neural Network (TaijiGNN), which combines cycled GAN and ancient Chinese "Taiji" philosophy. TaijiGNN consists of a pair of generators performing in opposite directions. The output of one generator is connected to the input of the other one, forming a loop structure. This loop structure can pass the input through the output domain and then back to the input domain. Therefore, TaijiGNN can convert the problem of comparing images in different domains into the problem of comparing images in the same domain. In the same domain, the severely under-constrained problem mentioned above can be solved naturally. Moreover, TaijiGNN has absorbed the essence of Taiji to train the pair of generators. During the training process, the pair of generators work like a couple, using their own advantages to complement each other, helping each other to achieve convergence, so that the entire system enters a state of dynamic equilibrium, which is very similar to Taiji's "Yin" and "Yang" bipolar working mode.

We use two classic spectral datasets, CAVE and ICVL, to evaluate VAE-GAN and TaijiGNN. And both of these two approaches use much less training data than state-of-the-art and reach or exceed their results. Moreover, the two approaches are implemented and verified on a heterogeneous computing platform designed using our hybrid machine learning methodology. Their training and inferencing speeds have been greatly improved.

Keywords: Machine Learning, Deep Learning, GPU Computing, FPGA Computing, High Performance Computing, Generative Adversarial Network, Variational Autoencoders, Multispectral Image, Spectral Reconstruction and Translation, Image Processing, VAE-GAN, TaijiGNN.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

LIST OF ABBREVIATIONS

AI	Artificial Intelligence
ALU	Arithmetic Logic Unit
CNN	Convolutional Neural Network
CPU	Central Processing Unit
DL	Deep Learning
DNN	Deep Neural Network
ETS	École de Technologie Supérieure
FGPA	Field Programmable Gate Array
GAN	Generative Adversarial Network
GPP	General Purpose Processor
GPU	Graphics Processing Unit
ML	Machine Learning
MSIs	Multispectral Images
PCIe	Peripheral Component Interconnect express
PUE	Power Usage Effectiveness
RGBs	RGB Images
TaijiGNN	Taiji Generative Neural Network
VAE	Variational Autoencoder

LIST OF SYMBOLS AND UNITS OF MEASUREMENTS

m	The number of training sample
x	Machine learning input
y	Machine learning output
z	Latent vector
α	Learning rate
θ	Machine learning parameter
μ	Mean
σ	Standard deviation
ϵ	Randomly sampled value from normal distribution
$N(0, I)$	Normal distribution
h_θ	Hypothesis function
$J_{(\theta_0,\theta_1)}$	Cost function
$\sum$	Sum

INTRODUCTION

0.1 Background and motivation

0.1.1 High performance machine learning hardware devices

Artificial intelligence (AI) is becoming one of the most significant technological development trends. Moreover, machine learning (ML) is the core of AI. The essence of ML is to let the computers discover the rules or the models beneath the data to conduct tasks without being explicitly instructed to do so (Wikipedia, 2021j).

Comparing with traditional programming, the most attractive character of ML is its data-driven property which can help us get a solution or to make a better decision without knowing well about the task's whole mechanism. Only some samples with labels are needed to train the model.

Based on the above advantages, ML has many applications, such as self-driving, recommendation engines, facial recognition. Some of the applications have been presented in Figure 0.1.

Although the MLs have many advantages and applications, their shortages or challenges are also apparent. Since most of MLs are data-driven approaches, more data often means higher performance. Many experts have verified this viewpoint, such as Andrew NG's has explained this phenomenon in his book "Machine learning yearning" (Ng, 2017). The training data and the model's performance have the relationship as Figure 0.2 shown.

Moreover, machine learning often consists of two phases, the training phase, and the inferencing phase. These two phases, especially the training phase, involve much high-density computing work like matrix operations (Figure 0.3).

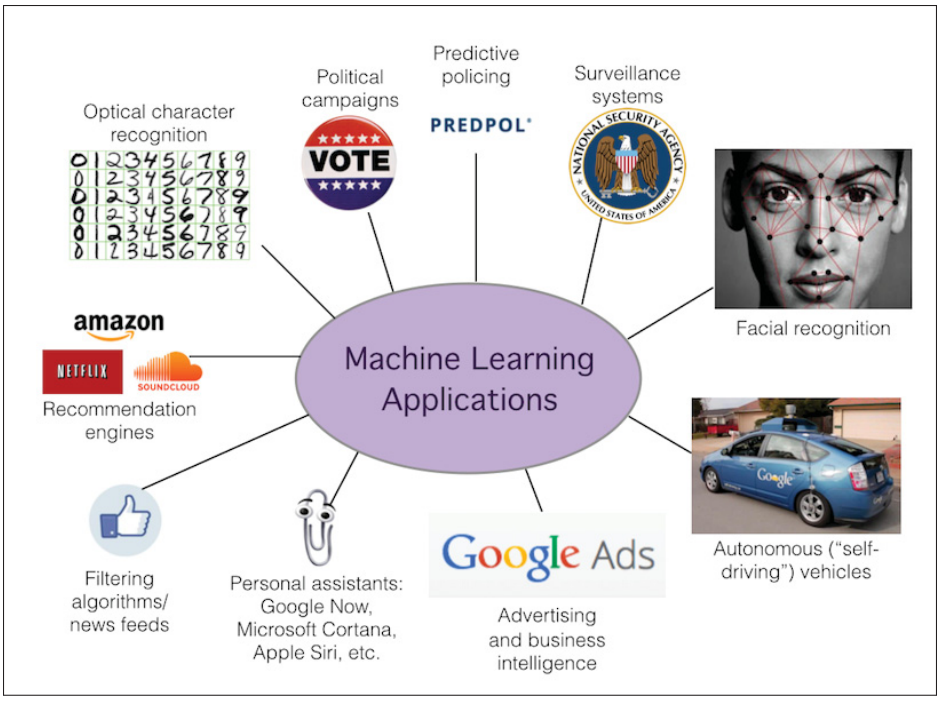

Figure 0.1 Machine learning applications
Taken from Marcos et al. (2021, Website)

More data inevitably bring in more processing and computing requirements. However, general-purpose processors like CPUs, which execute the instructions in sequence, are hard to meet the MLs' high-density computing requirements.

Currently, from consumer-grade computers to high-end business servers, the GPPs still are the dominators. Figure 0.4 presents the GPP's standard architecture. From the figure, we can find, most areas of the GPP consist of the control unit. The arithmetic logic unit (ALU) only takes a small area. This specific architecture determines the GPP is good at complex logical control but poor at numerical calculation.

Moreover, the sequence executing instruction system is another disadvantage of the GPP. As Figure 0.5 shown, the traditional CPU executes the instruction one by one, which becomes the bottleneck during training the ML's models.

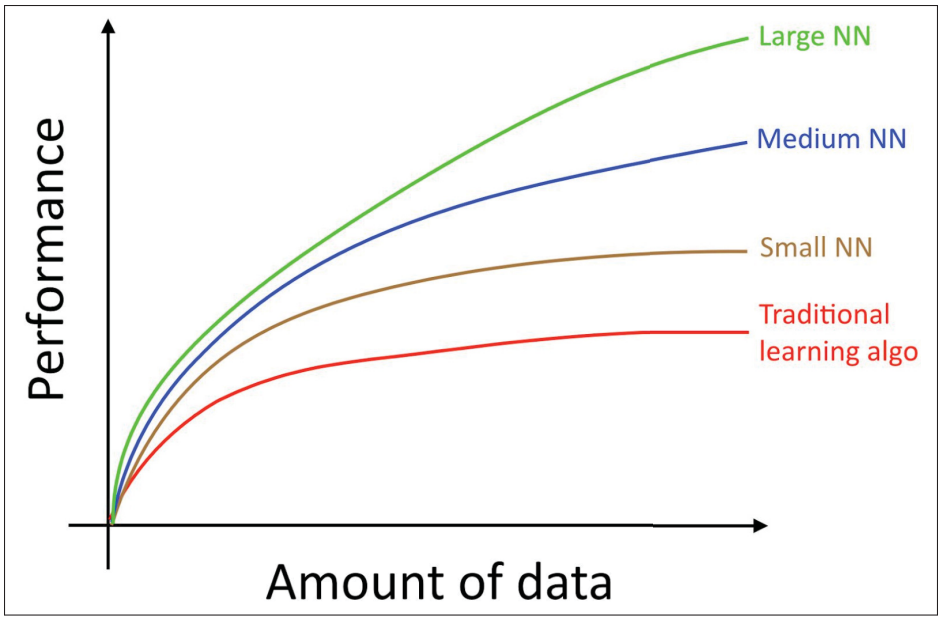

Figure 0.2 The relation between the model performance and the amount of data
Taken from Ng et al. (2021, Website)

How to improve the efficiency of machine learning becomes the most significant motivation of our research. Since the architecture of GPP is the main bottleneck causing low-performance machine learning, we have to find some other substitution hardware devices with different architectures to overcome this shortage.

The full name of GPUs is Graphics Processing Units. In the first, they have been designed to assist CPUs to process images and videos. However, with machine learning booming, some researchers found GPUs are also perfect for general computing jobs like machine learning and got a surprising effect.

All of the GPU's high-performance computing powers are from their particular architectures. We use Figure 0.6 to demonstrate the differences between CPU and GPU. From the figure, we can find that the GPU has much more computing unit-ALUs than the GPU has. These ALUs

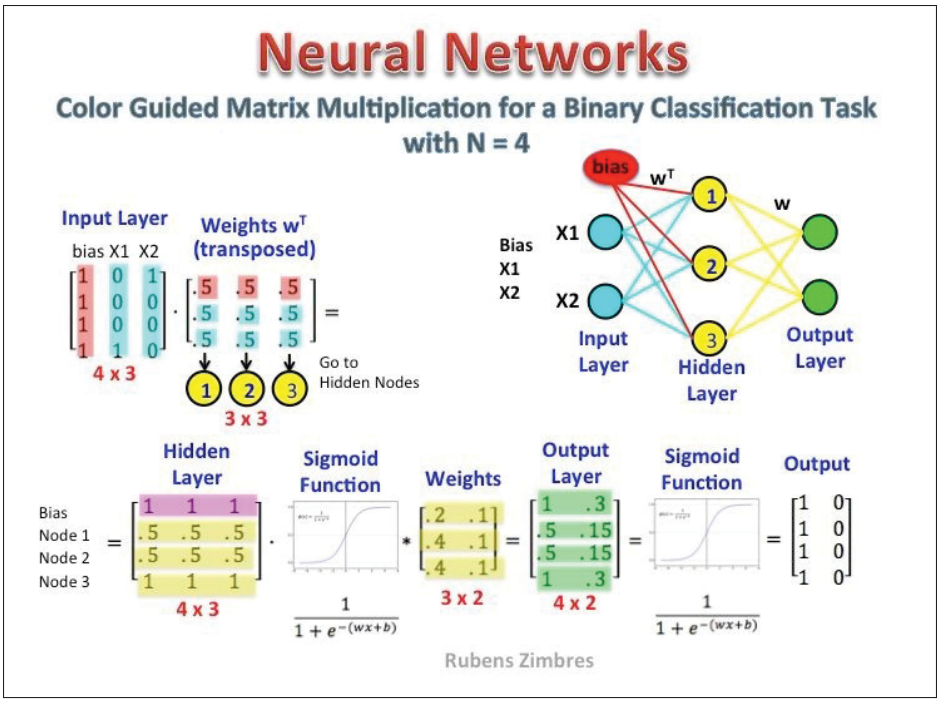

Figure 0.3 The complex operation and computing of machine learning
Taken from Zimbres et al. (2016, Website)

would be well programmed to execute large-scale parallel computing, making the GPUs become idea machine learning accelerating devices.

In addition to GPU, Field Programmable Gate Arrays (FPGAs) are also suitable to accelerate machine learning. Figure 0.7 is the architecture of FPGA.

It consists of many configurable logic cells, each of which can be configured to a specific functional circuit such as Adder or Multiplexer (MUX). We can use hardware description language (HDL) to implement different functions on the FPGA. For example, like GPU, we can implement many ALUs on the FPGA to execute instructions in parallel, as shown in Figure 0.8. Besides, unlike GPUs and CPUs, we can squeeze the hardware acceleration capability by optimizing the system architecture design in a finer-grained level according to the specific job, which is the prominent character only the FPGAs have.

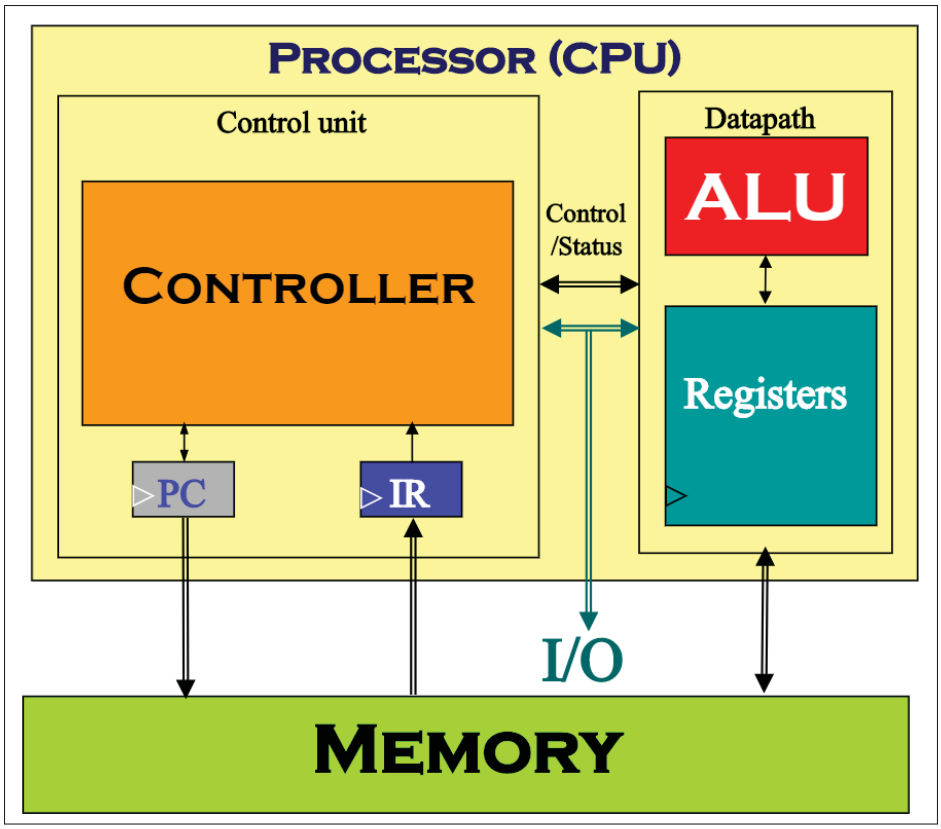

Figure 0.4 The General purpose processor's architecture
Taken from Robi Polikar et al. (2014, Website)

Until now, we have found two kinds of promising hardware devices, GPUs and FPGAs, which may accelerate the machine learning processes in a highly efficient way. In the following subsection, the thesis will introduce several applications of high performance machine learning. These applications could also evaluate the above hardware devices' accelerating performances.

0.1.2 The applications of high performance machine learning

Almost all ML algorithms require high-density computing power. After carefully comparing and analyzing, we found that ML algorithms related to image processing like CNN are best to evaluate the hardware accelerating performance. Since images often consist of tens of millions of

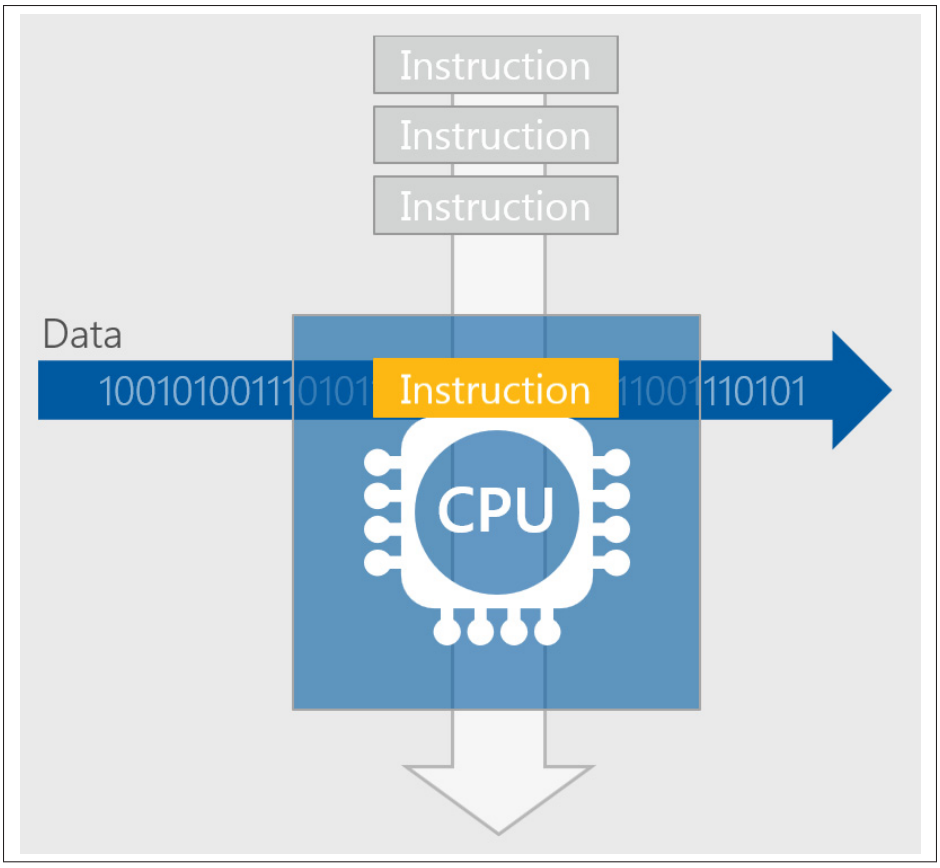

Figure 0.5 The instruction executing way of GPP
Taken from Russinovich et al. (2017, Website)

pixels, most of which have to be processed individually, image processing unavoidably involves much high-density computing work.

Therefore, we chose two image processing application scenes as our primary research topics: handwritten digit recognition and spectrum translation. Since the handwritten digit recognition algorithm has been studied well, we only focus on enhancing its training and inferencing performance based on the GPU and the FPGA.

Spectrum translation consists of two directions: translating RGB images to multispectral images and the reverse. A multispectral image is obtained by capturing image data within specific wavelength ranges across the electromagnetic spectrum (Wikipedia, 2020d). Multispectral

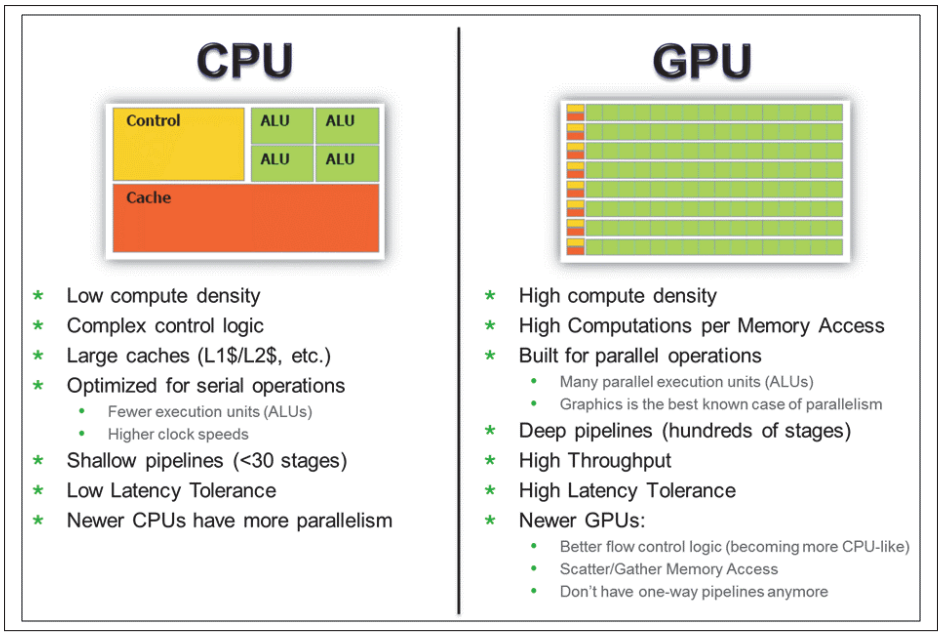

Figure 0.6 The differences between CPU and GPU
Taken from Rowe et al. (2017, Website)

images have broad application fields, such as space-based imaging, military target tracking, land mine detection, and farming (Photography Course, 2021).

Figure 0.9 is a multispectral imaging application in farming. From the figure, we could find that with the multispectral image's help, it is easy to manage the farming jobs, such as finding which block of the field needs more water and which block needs less fertilizer.

Although the multispectral images have many promising applications, high price and complex operation restrict their development. An easy way to think of is reconstructing the multispectral images from the RGB images. However, multispectral images contain much more information than RGB images. It is a severely underconstrained problem to reconstruct the ML images from the RGB images. Solving it is a big challenge, which I will give a detailed elaboration in the following sections.

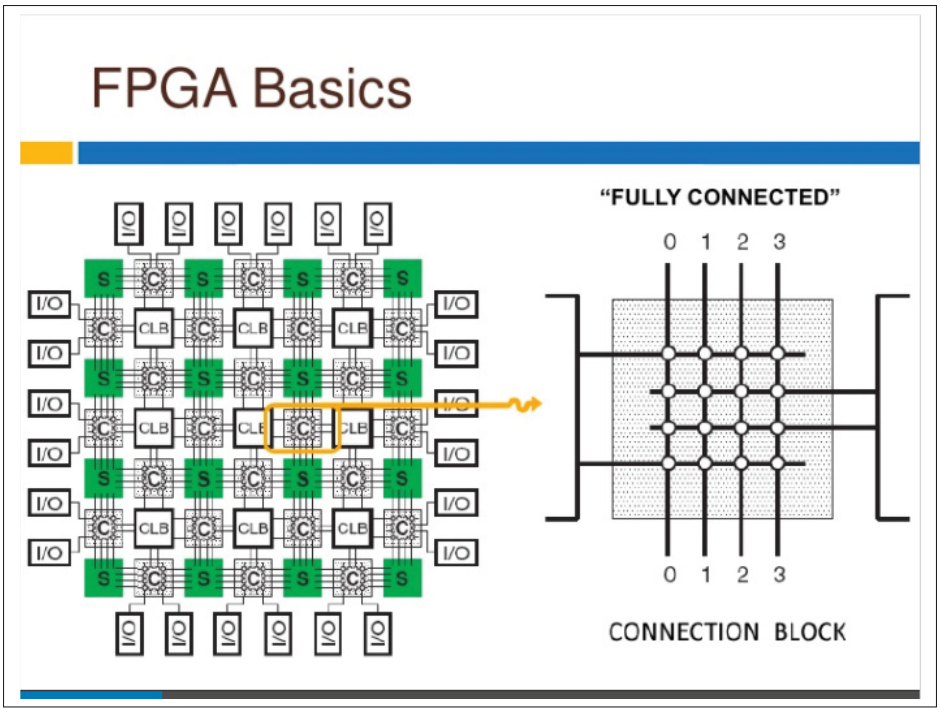

Figure 0.7 The architecture of FPGA
Taken from Wikipedia (2009, Website)

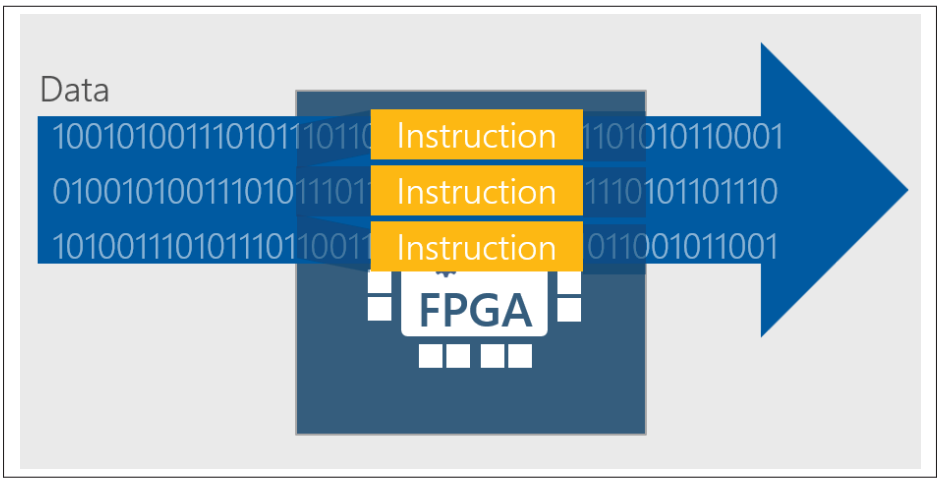

Figure 0.8 The executing instructions way of FPGA
Taken from Russinovich et al. (2017, Website)

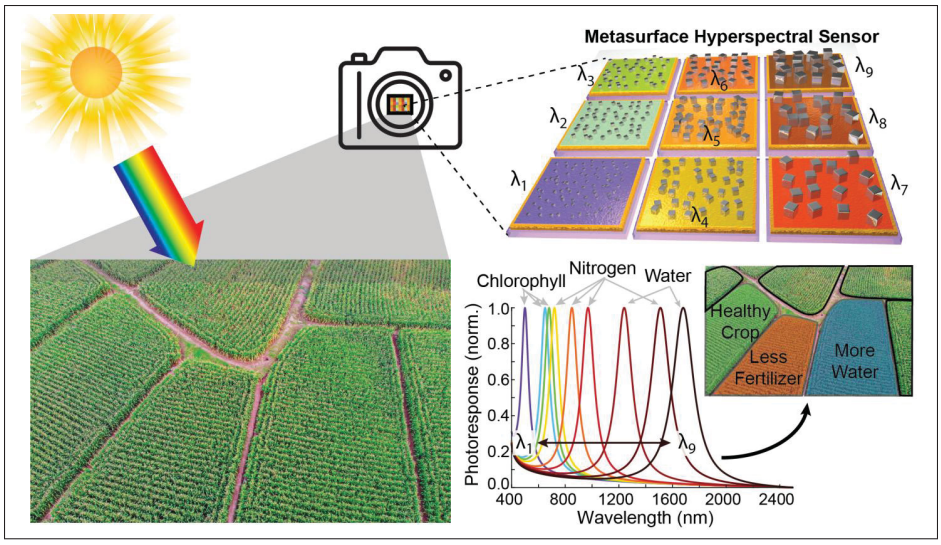

Figure 0.9 A multispectral application in farming
Taken from University et al. (2019, Website)

0.2 Problems

As described before, the bottleneck which blocks machine learning development much is the traditional hardware's low executing performance. We divided it into two subproblems to address this primary problem: One is to build a heterogeneous high-performance machine learning platform based on the GPUs and the FPGAs. The other is to implement different machine learning or deep learning algorithms, including spectrum translation, to evaluate the platform's performance.

0.2.1 High performance machine learning platform

Machine Learning or deep learning algorithms require high-density computing power. However, traditional general-purpose processors (GPPs), like CPUs, own little computing cores, and each core executes instructions in sequence, resulting in inadequate training and inferencing performance.

GPUs have many parallel computing cores, which could be exploited to accelerate ML or DL algorithms. However, currently, most of the algorithms are designed for sequence executing processors. So it becomes a significant challenge to flatten a sequence design into a parallel design to take the full advantages of the GPU parallel computing cores.

Since FPGAs' electronic logical circuits can be reconfigured, their designs are more flexible than GPUs and CPUs. We can custom the hardware architecture at a more refined grain level, such as the number of computing cores, the kind of memory, and the number of pipelines, which is both an opportunity and a challenge. The opportunity is we could squeeze the FPGAs' accelerating potential to the most. The challenge is it is too complex since it involved hardware and software coordinating design. Only when we coordinate design the hardware and the software well, the system can reach the highest performance.

Besides, different machine learning frameworks often have different model file definitions. Some frameworks like Google's Tensorflow did not open their model file definitions. So if we want to use their models, we have to crack the definitions. However, most machine learning models have more than four dimensions. Finding the correct order of the dimensions is not an easy job.

0.2.2 Spectral conversion

Since we have introduced it before, spectral conversions have the highest demand for computing power among all kinds of machine learning applications. But to do the spectral conversion, we have to solve the following two problems.

RGB images are formed by mixing different amounts of three fixed primary colors, red, green, and blue. On the contrary, multispectral images record different wavelength light luminosity distribution with different bands. Therefore, MSIs and RGBs have different synthesis mechanisms and belong to different image domains. How to do translation between two different domains with different definitions is a big problem.

Moreover, RGB images only have three bands. Otherwise, multispectral images often have tens or hundreds of bands. With the same spatial resolution, the multispectral images contain much more information than the RGB images. How to supplement the huge information gap between the RGB images and the MSI images is a significant challenge.

The above two problems are the main problems we have to solve while doing the bidirectional spectral conversions.

0.3 Contributions of the thesis

This thesis contributes to two main aspects. One aspect is proposing and implementing a new high-performance machine learning platform. The other is proposing two approaches (VAE-GAN and TaijiGNN) to solve the problems of spectrum translation.

0.3.1 High performance machine learing platform

To the best of my knowledge, previously, people have tried to use only GPUs or FPGAs to accelerate machine learning. The paper first proposed a novel GPUs and FPGAs hybrid architecture that can combine GPUs and FPGAs advantages to enhance machine learning performance.

Moreover, the paper gave a detailed design methodology and implemented two machine learning algorithms to evaluate the platform's performance. The evaluations proved that this kind GPU-FPGA based machine learning platform has a higher performance than any uni-GPU or uni-FPGA platform. Comparing with the uni-GPU designs, it has lower power consumption and delay. And Contrasting to the uni-FPGA implementation, its development difficulty is much lower.

0.3.2 Spectral conversion

The main contributions of this part are the two novel approaches (VAE-GAN and TaijiGNN) in the spectral conversion aspect. Both of these two approaches can solve the severely under-constrained problems while reconstructing the spectral images.

0.3.2.1 VAE-GAN

Most traditional neural networks are dependent, which means one input maps to one output. The dependent networks fail to solve the one input mapping to many output problems since they cannot create new patterns or distributions.

The VAE-GAN innovatively combines VAE and GAN in a network to solve one input many outputs problems, including the spectrum-reconstruction problem. The approach breaks the classical autoencoder's latent vector and re-parameters the latent vector with a stochastic variable randomly sampled from the normal distribution, then leverages the GAN to train the generator to produce MSI-like images.

Besides, the VAE-GAN found the best inserting noise place that is in the middle of the autoencoder. It is significant to insert the noise into the right place. Because if we insert the noise in other places like the input side, the noise is easily ignored by the network and cannot affects the outputs.

0.3.2.2 TaijiGNN

The most significant contribution of TaijiGNN is it merges the CycleGAN's cycle architecture and the ancient Chinese philosophy Taiji's complementary concept to create a new neural network. TaijiGNN can turn the under-constrained spectral conversion problem from comparing the two different domain images to the same domain images. In the same domain, no matter

which kinds of conversions are easy to accomplish. And the two biggest spectrum translation challenges can be solved naturally.

Moreover, TaijiGNN's application fields are not limited to spectrum translation. It has broad applicational fields such as image and label translation.

Furthermore, unlike other related work, TaijiGNN can do the bi-directional spectral conversion simultaneously.

0.3.3 List of publications

Focusing on the above topics, I have published a total of three journal papers, three conference papers, and a U.S. patent, which have been listed as follows.

0.3.3.1 Journal papers published

1. X. Liu, HA. Ounifi, A. Gherbi, W. Li and M. Cheriet, "A hybrid GPU-FPGA based design methodology for enhancing machine learning applications performance", in Journal of Ambient Intelligence and Humanized Computing (JAIHC), vol. 11, pp. 2309–2323, 2020, doi:10.1007/s12652-019-01357-4. (Liu, Ounifi, Gherbi, Li & Cheriet, 2019b)
2. X. Liu, A. Gherbi, Z. Wei, W. Li and M. Cheriet, "Multispectral Image Reconstruction From Color Images Using Enhanced Variational Autoencoder and Generative Adversarial Network", in IEEE Access, vol. 9, pp. 1666-1679, 2021, doi: 10.1109/ACCESS.2020.3047074. (Liu, Gherbi, Wei, Li & Cheriet, 2021b)
3. X. Liu, A. Gherbi, W. Li, Z. Wei, and M. Cheriet, "TaijiGNN: A New Cycle-Consistent Generative Neural Network for High-Quality Bidirectional Transformation between RGB and Multispectral Domains", in MDPI Sensors, vol. 21, 2021, doi: 10.3390/s21165394. (Liu, Gherbi, Li, Wei & Cheriet, 2021a)

0.3.3.2 Conference papers published

1. Xu Liu, Hibat Allah Ounifi, Abdelouahed Gherbi, Yves Lemieux and Wubin Li, "A Hybrid GPU-FPGA-based Computing Platform for Machine Learning," The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018), Leuven, Belgium, 2018, vol. 141, pp. 104-111. (Liu, Ounifi, Gherbi, Lemieux & Li, 2018a)
2. Xu Liu, Abdelouahed Gherbi, Wubin Li and Mohamed Cheriet, "Multi Features and Multi-time steps LSTM Based Methodology for Bike Sharing Availability Prediction," The 14th International Conference on Future Networks and Communications (FNC-2019), Halifax, Canada, 2019, vol. 155, pp. 394-401. (Liu, Gherbi, Li & Cheriet, 2019a)
3. Hibat-Allah Ounifi, Xu Liu, Abdelouahed Gherbi, Yves Lemieux and Wubin Li, "Model-based Approach to Data Center Design and Power Usage Effectiveness Assessment," The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018), Leuven, Belgium, 2018, vol. 141, pp. 143-150. (Ounifi, Liu, Gherbi, Lemieux & Li, 2018)

0.3.3.3 Patents

1. Xu Liu, Wubin Li, Yves Lemieux, Abdelouahed Gherbi, Hibat-Allah Ounifi, "METHOD, APPARATUS AND SYSTEM FOR HIGH PERFORMANCE PERIPHERAL COMPONENT INTERCONNECT DEVICE RESOURCE SHARING IN CLOUD ENVIRONMENTS," Publication Number: WO/2020/245636, International Application No. PCT/IB2019/054737, International Filing Date 06.06.2019.

0.4 Organization of the thesis

This thesis is based on three published journal papers, and the three journal papers are presented in Chapters 3, 4, and 5, respectively.

The remaining chapters of the thesis are composed as follows: In the first, Chapter Introduction introduces the thesis' background, motivation, problems, and contributions.

And Chapter 1 gives a literature review about related work.

After that, Chapter 2 elaborates the thesis' objectives and general methodology.

In Chapter 3, the first journal paper, "A Hybrid GPU-FPGA based Design Methodology for Enhancing Machine Learning Applications Performance" is presented.

And "Multispectral Image Reconstruction From Color Images Using Enhanced Variational Autoencoder and Generative Adversarial Network" is shown in Chapter 4.

After that, the third paper, "TaijiGNN: A New Cycle-Consistent Generative Neural Networks for High-Quality Bidirectional Transformation Between RGB and Multispectral Domains" is shown in Chapter 5.

And Chapter 6 explained the detailed relation among the three journal papers.

Finally, Chapter Conclusion concludes the whole thesis and recommends some future work.

CHAPTER 1

LITERATURE REVIEW

1.1 Machine learning

1.1.1 What is machine learning

The earliest definition from Arthur Samuel is "Machine Learning: Field of study that gives computers the ability to learn without being explicitly programmed". (Samuel, 1959) This definition seems not so professional.

Tom Mitchell gave a more professional definition. That is "Well-posed Learning Problem: A computer program is said to learn from experience E with respect to some task T and some performance measure P, if its performance on T, as measured by P, improves with experience E". (Mitchell *et al.*, 1997)

It is still very obscure to understand. Now let's use a concrete example to explain its meaning. Supposing your email program watches which email you mark or do not mark as a spam email, and based on that, learns how to filter spam better. During this example, "Classifying emails as spam or not spam" is "T". "Watching you label emails as spam or not spam" is "E". "The number (or fraction) of emails correctly classified as spam/not spam" is "P".

In my opinion, machine learning is a process that abstracts as much information as possible from datasets to get a prediction model.

1.1.2 The brief history of machine learning development

Early in 1950, Alan Turing invented the "Turing Test" to decide if a machine has human intelligence. A machine needs to mislead a person into thinking it is also human to pass the exam. (Turing, 2009)

In 1952, Arthur Samuel developed the first machine learning program. It was a computer checker run on an IBM computer. The program can be improved by learning what kind of moves constitute the winning policies and combining them into the program. (Samuel, 1959).

And in 1957, Frank Rosenblatt created the first perceptron neural network, which can mimic the human brain thinking processe. (Wikipedia, 2021c).

In 1967, T. Cover et al. proposed the "nearest neighbor" algorithm. Their approach allows machines to own fundamental pattern recognition functions, which could be practiced to plan a route for traveling salespeople. Even they start from a different city but guaranteeing they arrive all cities in the shortest journey. (Cover & Hart, 1967).

In 1979, Stanford University students created a "Stanford Cart," which can drive and evade obstacles in a flat independently. (Earnest, 2012).

In 1981, Gerald Dejong et al. proposed Explanation Based Learning (EBL). EBL can learn from a single training example and produce a general rule by dropping unimportant features. (DeJong & Lim, 2010).

In 1997, the Deep Blue of IBM first defeated the human world champion at chess.

In 2006, Geoffrey Hinton invented deep learning neural network to recognize objects and words in videos and images. (Wikipedia, 2021d).

In 2010, Microsoft created a human motion tracking device Kinect, which can collect human motion features at a rate of 30 times/s, enabling humans to communicate with the PC through actions and indications. (Wikipedia, 2021g).

And in 2011, Google built Google Brain, whose deep neural network can learn to identify and classify objects and reach a cat brain level. (Wikipedia, 2021e).

In 2012, Google X Lab revealed an ML algorithm that can recognize cats in Youtube videos. (Wikipedia, 2021f).

In 2014, Facebook disclosed DeepFace, an ML algorithm that can identify or distinguish people on images to human level. (Wikipedia, 2021b).

In 2015, Amazon launches machine learning as a service on their AWS platform. (Amazon Inc., 2021).

In 2016, Google Deep Mind created AlphaGo, which defeats a world-class Chinese board game Go player. The board game Go is much more complex and more complicated than chess. (Wikipedia, 2021a).

From the above history, we may find the machine learning algorithm develop slowly before 2010. But after 2010, the speed of machine learning development accelerates. One of the most important reasons is the acceleration hardware like GPUs made significant progress. People can quickly and inexpensively get acceleration hardware to speed up their machine learning algorithms.

1.1.3 Main machine learning algorithms

Figure 1.1 demonstrates the types of machine learning algorithms. According to the label's existence, ML algorithms can be divided into supervised learning and unsupervised learning. ML algorithms can also be categorized into traditional ML and artificial neural networks depending on if they have neural networks. The following sections will give a detailed introduction to supervised learning, unsupervised learning, and artificial neural network.

1.1.3.1 Supervised machine learning

Supervised machine learning often has labeled training data. That means every training sample has a mapping ground truth. We can use these mapped datasets to train the machine learning models.

I will use a concrete example to explain supervised learning's basic concepts and training flow.

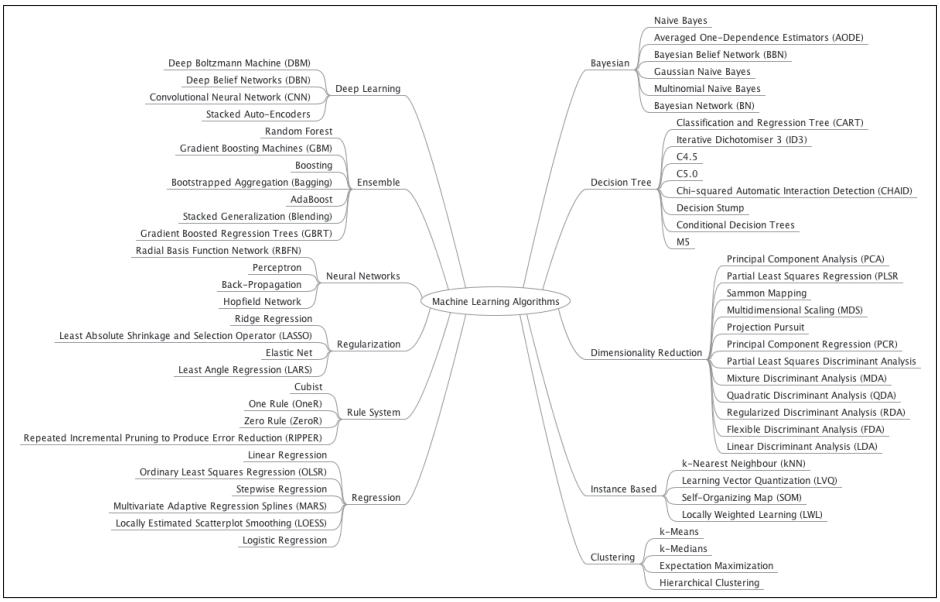

Figure 1.1 Types of machine learning algorithms
Taken from Rokad et al. (2019, Website)

Suppose we have a house price data table as Figure 1.2 presenting. "m" stands for the number of training samples.

Training Set

Size in feet² (x)	Price ($) in 1000's (y)
2104	460
1416	232
1534	315
852	178
...	...

m = 47

Figure 1.2 The house price dataset
Taken from NG et al. (2018, Website)

Figure 1.3 is general supervised learning flow. We input the training set into the learning algorithm and get a hypothesis function. And then, we input the x (Size of the house) into the h (hypothesis function) to get y (Estimated price). During this flow, the most crucial part is the hypothesis function. So the critical aim of machine learning is to get the hypothesis function. The hypothesis function is a function you suppose is similar to the actual function. It usually has

many different shapes. We give the first hypothesis according to experience but will adjust the hypothesis according to test results.

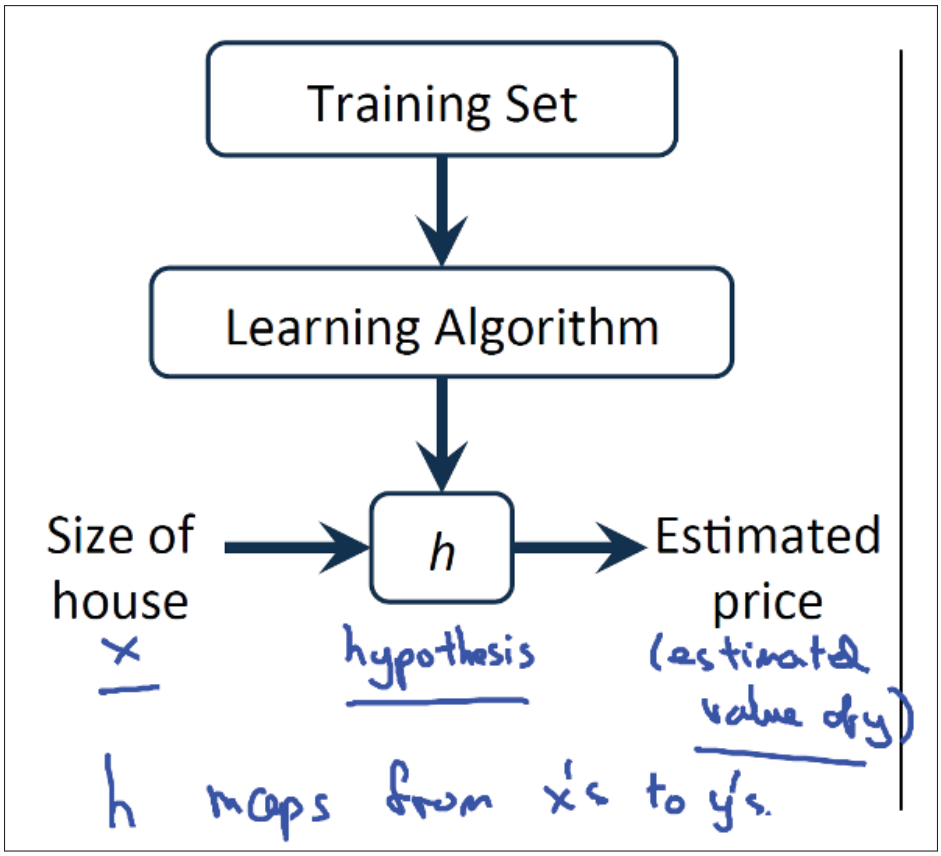

Figure 1.3 The supervised learning flow
Taken from NG et al. (2018, Website)

In this housing price example, we suppose the hypothesis is as Equation 1.1:

Hypothesis:

$$h_\theta = \theta_0 + \theta_1(x) \tag{1.1}$$

θ_i stands for the parameters, which is the machine learning target. To get the θ_i, we need to calculate the cost function.

Equation 1.2 indicates for the cost function.

Cost function:

$$J(\theta_0, \theta_1) = \frac{1}{2m} \sum_{m}^{i=1} (h_\theta(x^i) - y^i)^2 \tag{1.2}$$

"m" is the number of training samples. The cost function is a way to measure the differences between the hypothesis and the actual result. Here we use the average square to calculate the cost.

Our goal is to minimize J by changing (θ_0, θ_1)'s value.

Goal:

$$Minimize_{\theta_0,\theta_1} J(\theta_0, \theta_1) \tag{1.3}$$

The θ_is which make J minimal are we want to get. And then, we can input these θ_is into the hypothesis function to predict new data.

There are many ways to minimize cost function J, but Gradient Descent is most frequently used.

Algorithm 1.1 Gradient descent algorithm

```
1 while Not convergence do
2 |   θ_j := θ_j − α ∂J(θ_0,θ_1)/∂θ_j (for j = 0 and j = 1);
3 end while
```

The α is called the learning rate. If α is too small, gradient descent can be slow. If α is too large, gradient descent can overshoot the minimum. It may fail to converge or even diverge. We should adjust the learning rate according to the learning effects.

1.1.3.2 Unsupervised machine learning

Unsupervised machine learning is to infer a model to describe hidden structure from "unlabeled" data sets. The critical point of unsupervised machine learning is that we don't know the ground truth for each example. Except owing the data, we don't know what kind of relations and

classifications are inside the data. However, these relations and classifications are what we want to get. Figure 1.4 shows the differences between supervised learning and unsupervised learning.

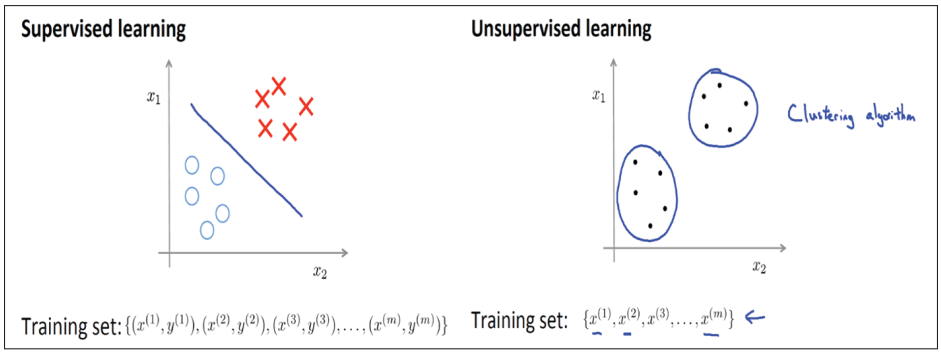

Figure 1.4 The supervised learning VS the unsupervised learning
Taken from NG et al. (2018, Website)

In supervised learning, every training sample has input data x and paired output data y. And there is a clear boundary between them for classification. However, in an unsupervised learning training set, there is only x, no paired data y. So we don't know the distance from the actual result when we use x to make a prediction. Also, there is no fixed boundary between them, no fixed classification. Therefore, clustering algorithms become a better solution for classifying.

Figure 1.5 presents several clustering applications, such as market segmentation, social network analysis, organize computing cluster, and astronomical data analysis.

1.1.3.2.1 K-mean clustering algorithm

K-mean is one of the most famous clustering algorithms. It aims to distribute n observation points into k clusters. Every observation point related to the cluster has the nearest mean in each cluster, which is the basic concept of the cluster (Wikipedia, 2021h). The K-means algorithm is as follows.

Figure 1.6 shows the whole process of the K-mean algorithm. In a word, the goal of K-means is

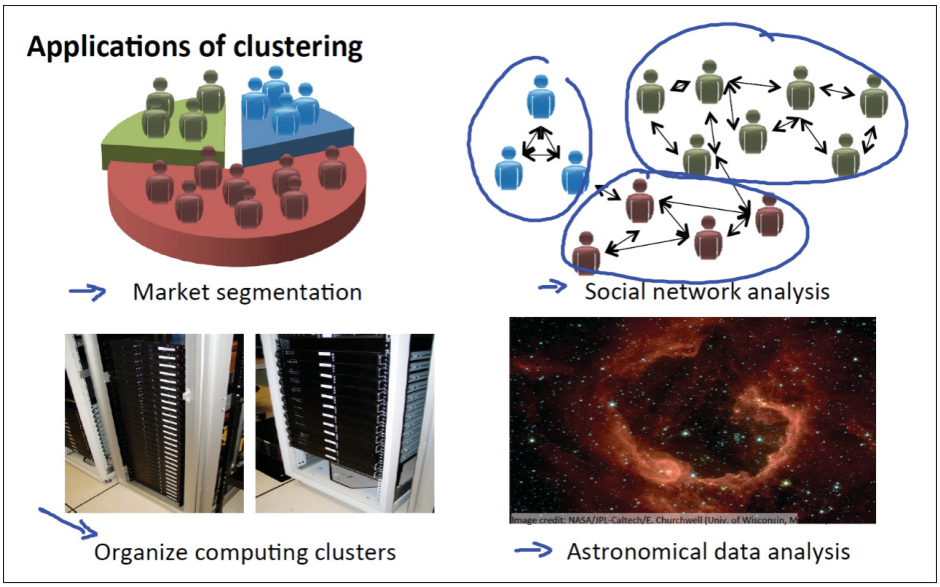

Figure 1.5 The clustering applications
Taken from NG et al. (2018, Website)

Algorithm 1.2 K-means algorithm

```
Randomly initialize K cluster centroids μ_1, μ_2, ..., μ_k ∈ ℜ^n;
while True do
    for i = 1 to m do
        c^(i) := index(from 1 to k) of cluster centroid closest to x^(i);
    end for
    for k = 1 to k do
        μ_k := average(mean) of points assigned to cluster k;
    end for
end while
```

to find every cluster's centroids and make the sum of the distance between every centroid and points minimal.

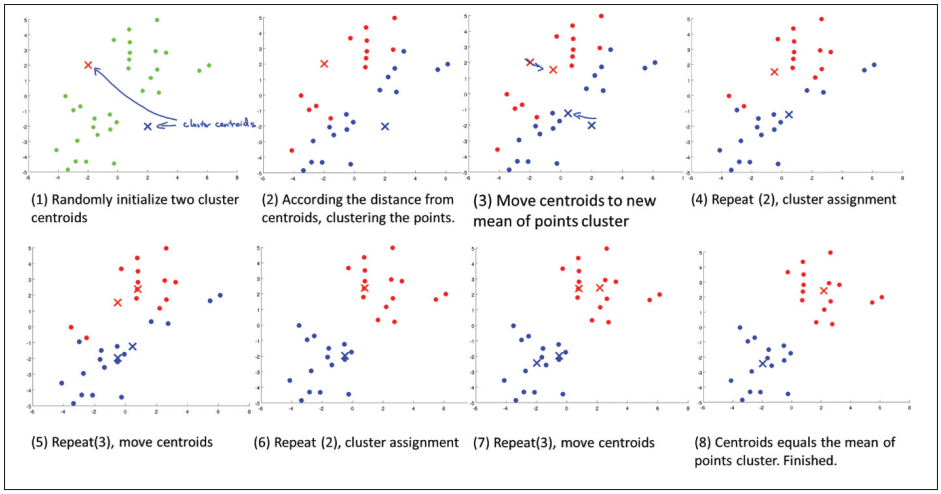

Figure 1.6 The process of K-means algorithm
Taken from NG et al. (2018, Website)

1.1.3.3 Artificial neural network

The term "Artificial Neural Network" comes from biological neural networks that constitute animal brains (Figure 1.7). Such systems learn (progressively improve performance) to do tasks by considering examples, generally without task-specific programming. (Wikipedia, 2021i)

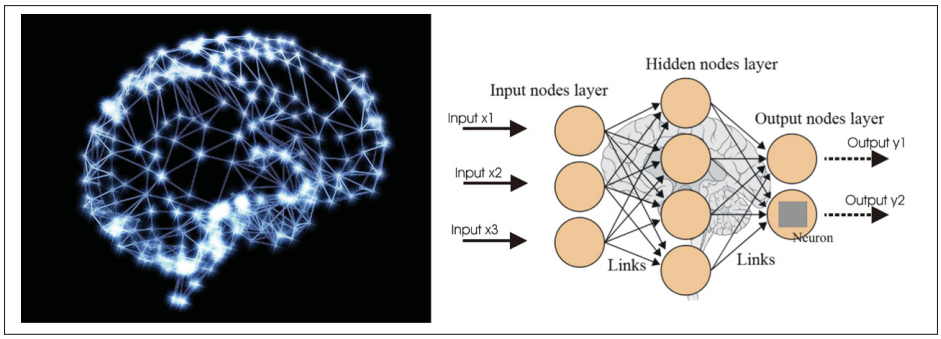

Figure 1.7 The biological neural network VS artificial neural network
Taken from MUKHERJEE et al. (2017, Website)

The basic structure of neural networks has at least three layers: input layer, hidden layers, and output layer. The input layer and output layer are only one layer. Hidden layers may have several kinds of layer combinations depending on the computing requirements.

In a word, the neural network uses many layers' activations to feel(calculate) outside inputs, compared with before experiences(labels), and gives the last results(outputs).

Since the thesis involved GAN and CycleGAN, I will introduce them in the following paragraphs.

1.1.3.3.1 Generative adversarial network

Figure 1.8 presents the classical structure of the generative adversarial network. The classical GAN architecture often contains two neural networks: generator (G) and discriminator (D). In detail, The target of G is to synthesize fake samples and try to make D believe that the counterfeits are real. The goal of D is to distinguish the synthesized samples from the real samples. The competition drives G and D to continuously optimize their parameters until the counterfeits are indistinguishable from the genuine ones (Goodfellow *et al.*, 2014). Equation (1.4) (Goodfellow *et al.*, 2014) is the total value function of the GAN architecture, which includes the sub-value function (1.5) of the discriminator and the sub-value function (1.6) of the generator. Among them, $p_{data}(x)$ is the distribution of real data x and $p_z(z)$ stands for the distribution of random noise. We can find that G and D act like two players playing the minimax games from the value functions. Specifically, D is trying to maximize its strength of discrimination. Oppositely, G is trying to improve the quality of counterfeits to minimize the difference from the real (Goodfellow *et al.*, 2014).

$$\min_G \max_D V(D,G) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1.4}$$

$$\max_D V(D) = \mathbb{E}_{x\sim p_{data}(x)}[\log D(x)] + \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1.5}$$

$$\min_G V(G) = \mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{1.6}$$

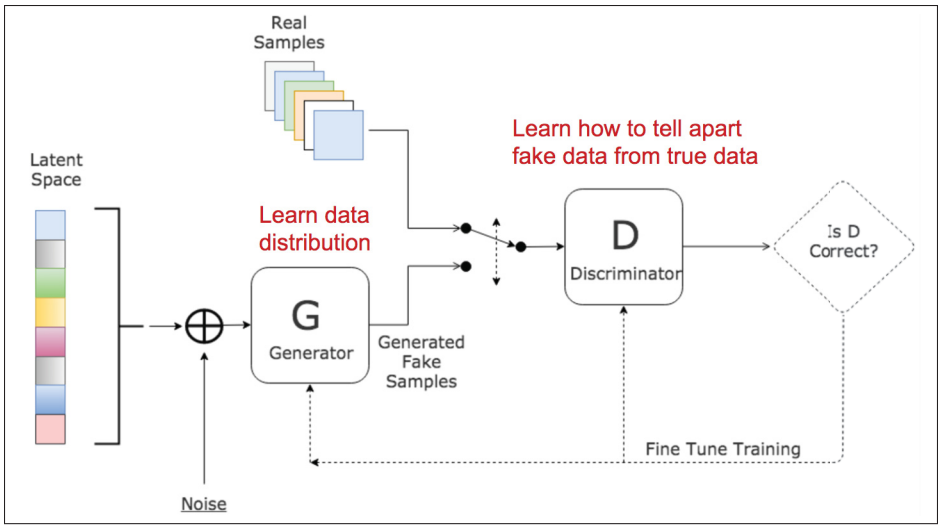

Figure 1.8 The classical structure of generative adversarial network
Taken from Gharakhanian et al. (2017, Website)

As the traditional GAN model only synthesizes fake samples from noise, it can only guarantee that the fake sample belongs to the target domain but can not specify which sample in the domain. However, sometimes, we hope we can specify the generated sample. For example, when we reconstruct the multispectral image from an RGB image, we hope to preserve every detail of the two images except the spectral or color channels. The traditional GAN can not achieve this goal. We need some improved GANs like CycleGAN.

1.1.3.3.2 CycleGAN

CycleGAN comes from GAN. It has an additional generator and an extra discriminator compared with the classical GAN. The two generators and two discriminators would form a loop, as shown in Fig. 1.9. X is the source domain, and Y is the target domain. G is a generator that translates X to Y, and F is another generator that converts Y to X. D_x is the X domain discriminator, which tells the probability that a sample came from the X domain rather than the Y domain. Similarly, D_y is also a discriminator for estimating the probability that a sample came from the Y domain rather than the X domain. The most innovative idea of CycleGAN is that it introduces two pairs

of cycle-consistency losses: $F(G(X)) \approx X$ and $G(F(Y)) \approx Y$, which will restrict the results of two generators G and F and make them converge faster(Zhu, Park, Isola & Efros, 2017).

Until now, the most successful application of CycleGAN is image-to-image translation like collection style transfer, object transfiguration, season transfer, photo enhancement, and so on (Zhu *et al.*, 2017). Inspired by the above applications, we found that CycleGAN is also suitable for performing bi-directional image translation between multispectral and RGB images. Compared with other existing methods (Nguyen, Prasad & Brown, 2014; Kaya, Can & Timofte, 2018), applying CycleGAN brings the following three advantages naturally:

1. Train the models with unpaired data sets.
2. Obtain two generators in one training process.
3. Need fewer data sets and achieve higher accuracy.

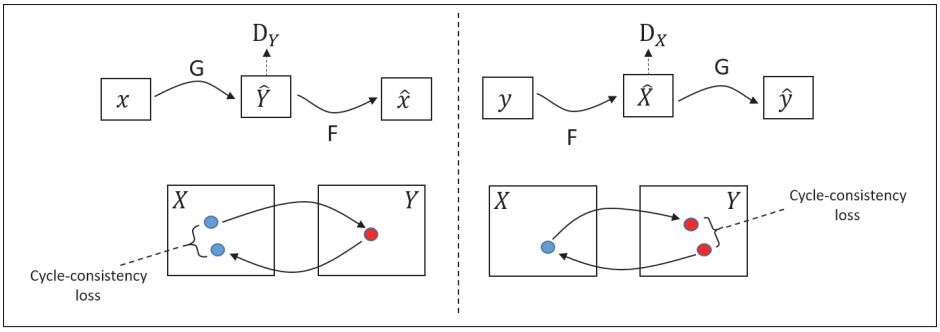

Figure 1.9 The classical structure of CycleGAN
Taken from Google (2020, Website)

1.2 High performance machine learning

With the machine learning complexity increasing, more and more persons are interested in improving machine learning performance with hardware.

Early in April 2016, Nvidia improved the performance and energy efficiency of the computation-demanding CNN. They reported, the Alexnet performance could reach 1150 imgs/s, 20 imgs/s/W and 2.2 TFLOPS on their GPU M4 (28nm). (Aydonat, O'Connell, Capalija, Ling & Chiu, 2017a).

In November 2016, Chen Zhang et al. proposed a uniformed convolutional matrix multiplication representation, which could improve both computation-intensive convolutional layer and communication-intensive fully connected (FCN) layer performance. Meanwhile, they designed Caffeine to maximize the underlying FPGA computing and bandwidth resource utilization, focusing on bandwidth optimization by the memory access reorganization. Meanwhile, they implemented Caffeine in the portable high-level synthesis and provided various hardware/software definable parameters for user configurations. Meanwhile, they integrated Caffeine into the industry-standard software deep learning framework Caffe. And they claimed their performance could reach 104 imgs/s, 4 imgs/s/W, and 365 GOPS on the Xilinx KU060 FPGA. And on the Xilinx Virtex7 690t FPGA, the performance could reach 181 imgs/s, 7 imgs/s/W, and 636 GFLOPS. (Zhang *et al.*, 2019).

In November 2016, Dong Wang et al. proposed an FPGA accelerator based on a new architecture of deeply pipelined OpenCL kernels. And they presented data reuse and task mapping techniques to improve the design efficiency. And their approach could reach 33.9 GFLOPS on Altera Stratix-V A7 FPGA. (Wang, An & Xu, 2016).

In December 2016, Roberto DiCecco et al. successfully integrated Xilinx SDAccel on the Caffe CNN framework, which could conduct CNN classification on the CPU-FPGA platforms. They implemented the Winograd convolution algorithm with their approach. And their results could reach 50 GFLOPS on a Xilinx Virtex 7. (DiCecco *et al.*, 2016).

In February 2017, Utku Aydonat et al. proposed a method that cached all intermediate features in buffers. Their approach could minimize the bandwidth of convolutional and fully connected layers and reached 1020 imgs/s, 23 imgs/s/W, and 1382 GFLOPS on Intel's Arria 10 FPGA. (Aydonat *et al.*, 2017a)

1.3 Spectral conversion

The RGB image base is the additive color model, in which red, green, and blue light are mixed in various ratios to reproduce all kinds of colors. Figure 1.10 is the CIE 1931 RGB color matching

function, which shows that we can get any wavelength light by composing different quantitative red, green, and blue lights. Furthermore, from Figure 1.11, we can find that with the three primary color points, we can get all kinds of colors inside of the triangle.

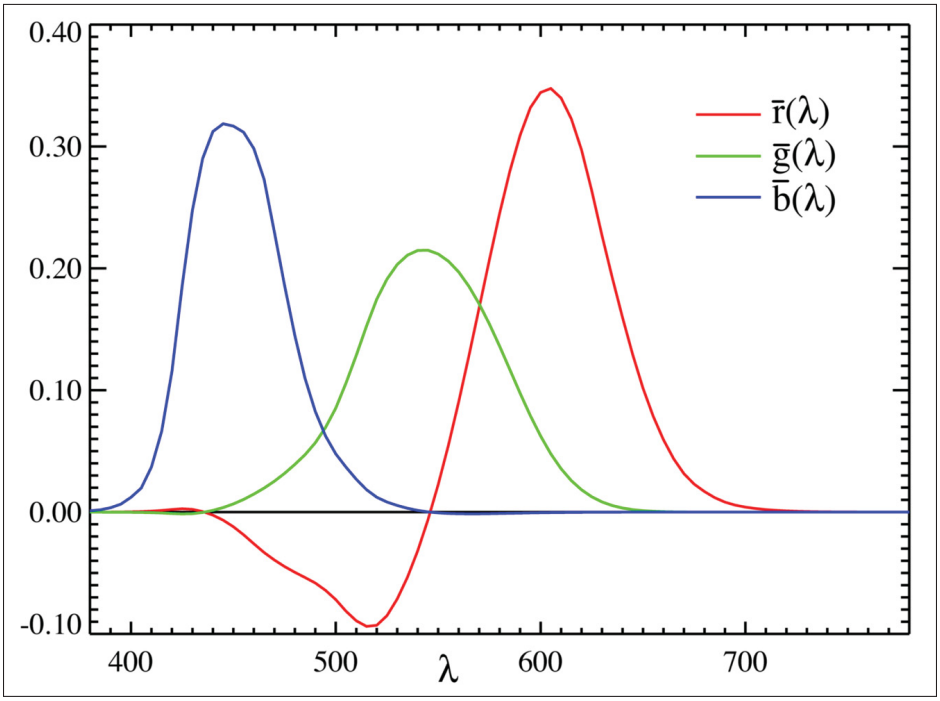

Figure 1.10 The CIE1931_RGB color matching function
Taken from Abraham et al. (2020, Website)

On the contrary, multispectral images are constituted of individual bands. And each band records the luminosity distribution of the specific wavelength light.

MSIs and RGBs have fundamental differences. They have no direct mapping relations. However, if we insist on using RGBs to represent the MSIs, we will meet a more tough problem-metamerism, which means one RGB pixel set may match many MSIs pixel sets. The main reason is the RGBs only have three bands, and MSIs have tens or more bands. The possible space of MSIs is much bigger than the space of RGBs. There is a massive information entropy gap between the two domains. Figure 1.12 demonstrates a concrete example, supposing there are a green motor vehicle and green leaves, we can not judge them only by their color. Otherwise, the MSIs can give us more spectral information to identify the objects.

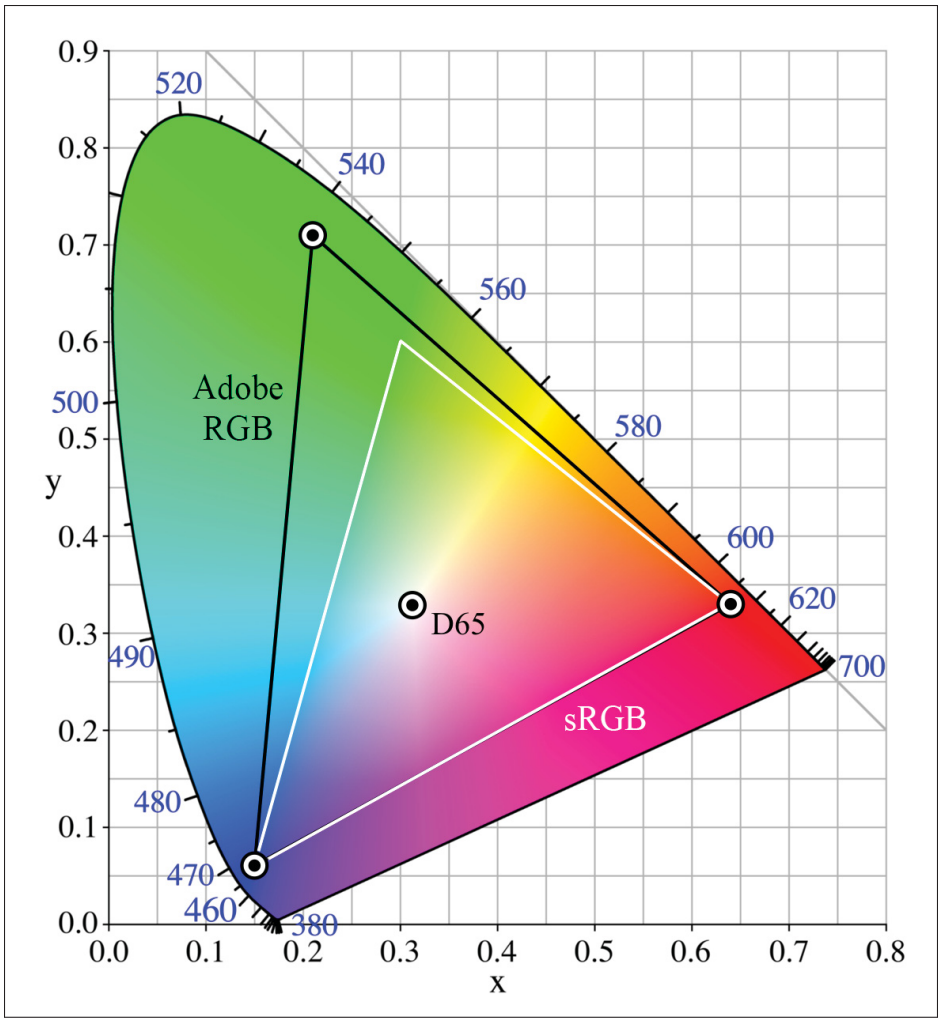

Figure 1.11 The chromaticity diagram
Taken from Abraham et al. (2020, Website)

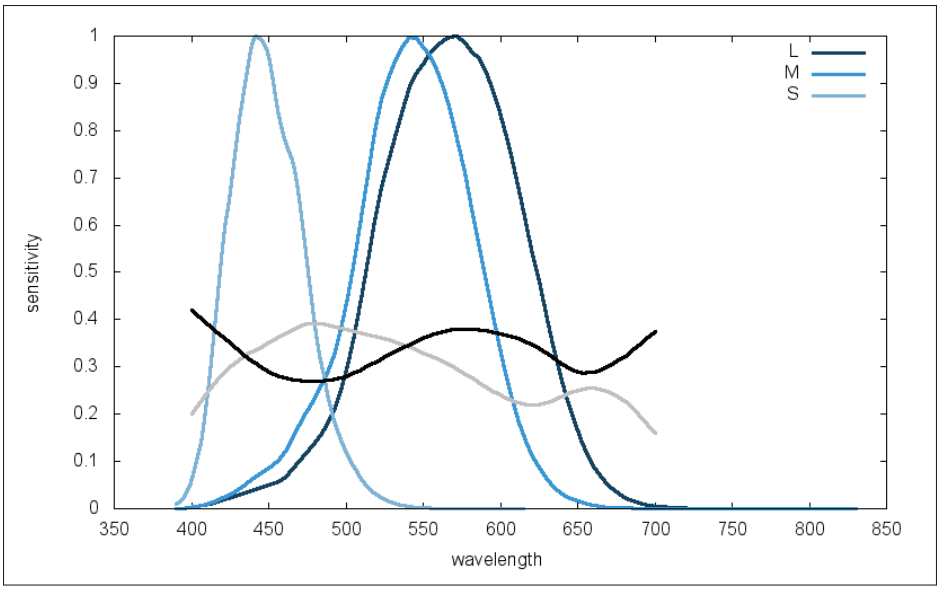

Figure 1.12 The metamerism problem
Taken from Abraham et al. (2020, Website)

CHAPTER 2

OBJECTIVES AND GENERAL METHODOLOGY

2.1 Main_objective

The main objective of this thesis is to explore a new type of high-performance heterogeneous machine learning platform and apply the platform to all kinds of machine learning applications to enhance their performance. Finally, we have to build a high-performance machine learning stack, as shown in Figure 2.1. The bottom layer of the stack is composed of a heterogeneous acceleration platform with GPU and FPGA as the main body. The top layer provides services for various machine learning applications. And this main objective will be fulfilled by the following two sub_objectives.

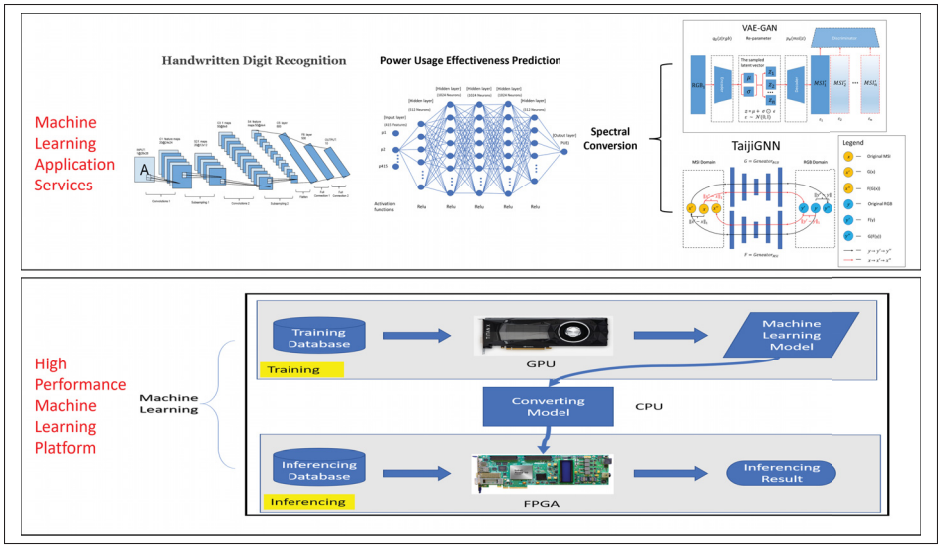

Figure 2.1 The High Performance Machine Learning Stack

2.1.1 Sub_objective: To build a high performance machine learning platform based on the GPU and the FPGA

This sub_objective aims to find the best GPU and FPGA composition tactic to build a high-performance machine learning platform, which can trade-off between the GPU's and FPGAs'

pros and cons to boost machine learning performance at most. Moreover, we anticipate this heterogeneous ML platform's performance is superior to any ML platform based on a single device like CPU, GPU, or FPGA.

2.1.2 Sub_objective: To apply the heterogeneous platform to machine learning applications

The second sub_objective is to improve the performance of various machine learning applications like handwritten digit recognition, power usage effectiveness prediction, and spectral conversion based on the first sub_objective's heterogeneous platform.

2.2 General Methodology

2.2.1 Build machine learning platform

Machine learning algorithms consist of two phases, training and inferencing. The training phase focuses on how to get the best model, which means the model's architecture and hyperparameters have to be adjusted often. Moreover, the amount of training data is generally huge, so that the training phase requires much more memory than the inferencing phase. Furthermore, each inferencing process is only one sample inputted to the forward network to get one output. On the contrary, each training process involves tens of thousands of forwarding and backward propagation computing. So the training phase demands more computing power than the inferencing phase.

Figure 2.2 is the architecture comparison of CPU, GPU and FPGA. From it, we could find GPUs' most crucial feature is vast memory and abundant parallel computing threads, which could exactly meet the above training phase's requirement for memory and computing power. Moreover, GPU development is as easy as CPU development, which has a mature development toolchain and environment. So the easy design way makes GPUs suitable for the changeable training algorithm.

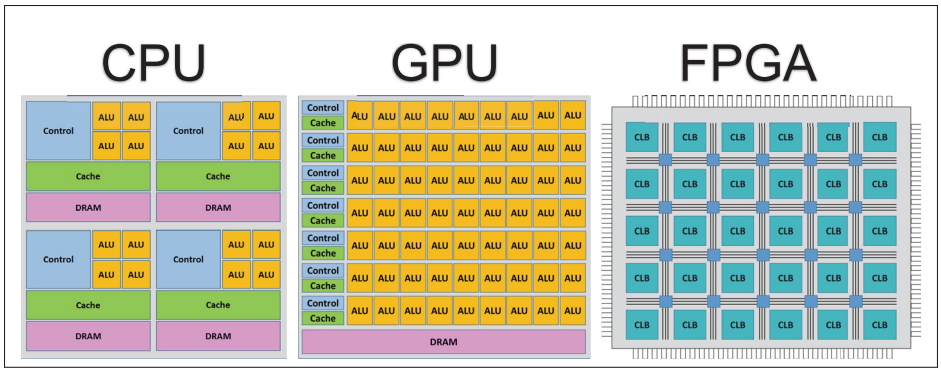

Figure 2.2 The architectures of CPU, GPU, and FPGA

On the contrary, FPGAs have tens of thousands of reconfigurable logic elements (LEs), which could be reconfigured into different function logic circuits like "AND," "OR," and "NOR" gates. And these LEs could be combined to form larger and more complex function logic circuits like arithmetic logic unit (ALU). So FPGAs' advantages are that we can custom the hardware logic circuits according to the requirement at a finer-grain level. In this way, we could maximally squeeze the FPGA's acceleration potential. Moreover, the FPGAs implement functions by hardware logic circuits, no need complex instruction system. So the FPGA delay time and power consumption are much less than GPUs and CPUs'. However, the FPGAs have shortages, like the long development period, complex development approach, and high price. During the inferencing phase, the algorithm is fixed, and the main goal is to pursue high performance. Therefore, the FPGAs are suitable for the inferencing phase.

Besides, the training framework and the inferencing framework are usually different, which means they have different model file formats. So the model generated from the training framework cannot be used directly on the inferencing framework. It needs a model converter between the training phase and the inferencing phase.

Figure 2.3 concludes the high performance machine learning platform's design methodology that is to implement the training phase on the GPUs, implement the inferencing phase on the FPGAs, and implement a model converter between the two phases.

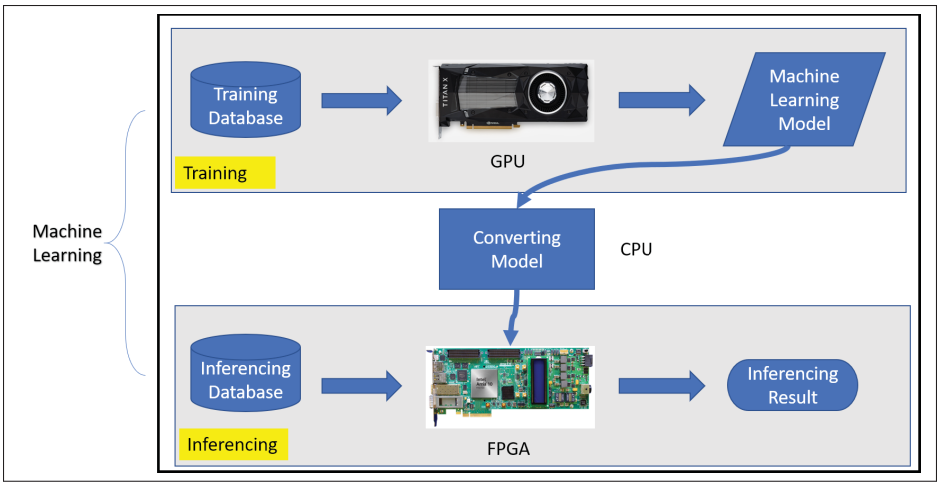

Figure 2.3 The architecture of heterogeneous machine learning platform
Taken from Liu et al. (2018, p. 104-111)

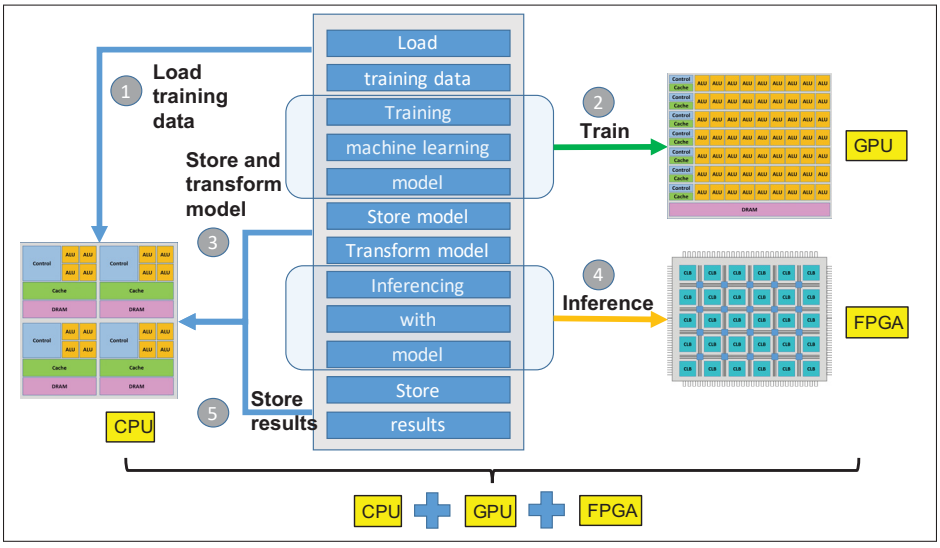

Figure 2.4 The detailed work flow of the heterogeneous machine learning platform
Taken from Liu et al. (2019, Website)

Figure 2.4 presents the workflow of the heterogeneous machine learning platform. The CPU loads the training data and sends the data and starting training signal to the GPU. Once the GPU finishes the training, the CPU fetches the model from the GPU, stores the model into storage,

and converts the model format to meet the inferencing requirements. Then the CPU sends the model, inferencing data, and starting inferencing signal to the FPGA to start inferencing. After inferencing, the CPU regains back the results and saves or uses them for tasks.

2.2.2 Machine learning platform applications

The ultimate goal of building a machine learning platform is to improve the performance of various machine learning applications. So several kinds of machine learning applications have been chosen to evaluate the platform.

2.2.2.1 Handwritten Digit Recognition

In this application, a LeNet-5 architecture was implemented on the GPU and FPGA, respectively. The GPU was used for training the model, and the FPGA was used for inferencing the model.

2.2.2.2 Power Usage Effectiveness Prediction

In this application, a multi-perceptron architecture was implemented on the GPU and FPGA, respectively. The GPU was used for training the model, and the FPGA was used for inferencing the model.

2.2.2.3 Spectral conversion

In this application, GPU was used for both training and inferencing. That was because currently, I focused on improving the spectral conversion algorithms and often tried different algorithms. It is much easier to implement an algorithm of the same scale using GPU than using FPGA. So I chose GPU to accelerate both the training and inferencing processes in the spectral conversion. In addition, because the training and inference processes use the same hardware architecture, no model conversion is required, so the model conversion module is eliminated.

Spectrum conversion's two biggest challenges are different domains (RGB domain and multispectral domain) have different definitions and how to supplement the huge information gap between the RGB images and the multispectral images. I proposed two approaches presented in the following subsections to address the above two challenges.

2.2.2.3.1 VAE-GAN

This approach combines Variational Autoencoder (VAE) and Generative Adversarial Network (GAN). The traditional Autoencoder consists of three parts: the encoder, the latent vector z, and the decoder. The VAE splits the latent vector z into the mean μ, and the standard deviation σ and then randomly samples a value ϵ from the normal distribution and re-parameters the latent vector z following Formula (2.1). Thus, one fixed latent vector could be re-parametered into unlimited latent vectors, which is the critical point of the VAE-GAN approach.

$$\begin{aligned} z &= \mu + \sigma \odot \epsilon \\ \epsilon &\sim \mathcal{N}(0, I) \end{aligned} \tag{2.1}$$

Since the bi-direction translations are symmetrical, we only take the process of reconstructing MSIs from RGBs as an example (Please refer to Figure 2.5):

1. One input RGB image will be encoded into a latent vector z. Then this vector will be re-parametered into unlimited latent vectors $z_1, z_2, ..., z_n$, which integrate the input RGB image information and the random variations. The vectors contained variations will be input into the decoder to get the corresponding MSI image.
2. The generated MSI images and the ground truth images will be input into the GAN's discriminator to train the GAN.
3. We get a generator, which can reconstruct MSIs from RGBs.

The process of reconstructing RGBs from MSIs is similar to the above process.

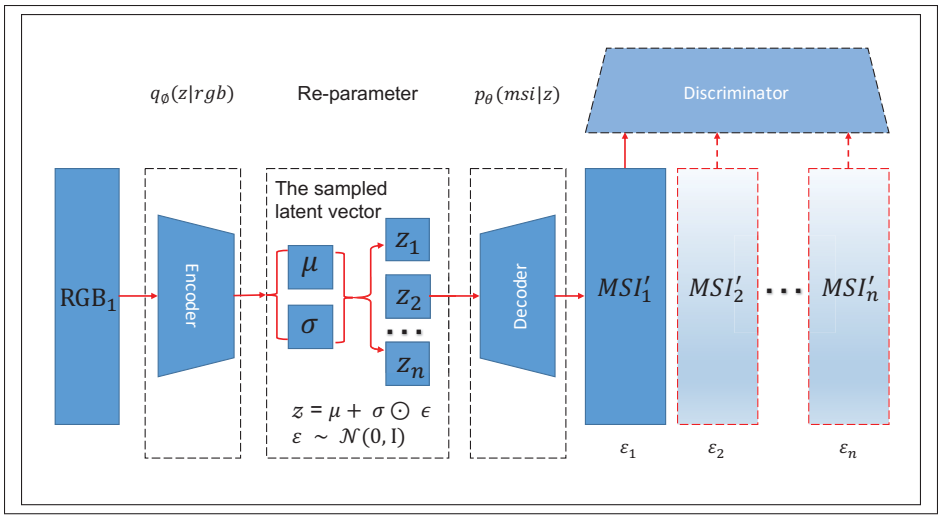

Figure 2.5 The detailed implementation of the VAE-GAN approach
Taken from Liu et al. (2020, p. 1666)

In summary, The VAE-GAN approach inserts stochastic variations into the classical Autoencoder. And the abstracted input information mixed with the stochastic variations can be used to train the GAN's generator to produce the output-like images. In this way, one input image could evolve several various latent vectors, and various latent vectors could map to various multispectral images. Thus one-to-many under-constrained problem is converted to many-to-many constrained problem. The two biggest spectrum translation challenges can be solved naturally.

2.2.2.3.2 TaijiGNN

TaijiGNN merges the cycle neural network and the ancient Chinese philosophy Taiji's concept to create a new neural network named "TaijiGNN."

From Figure 2.6, we could find that TaijiGNN consists of two generators, G_{MSI} and G_{RGB}. G_{MSI} is to reconstruct MSIs from RGBs, and G_{RGB} is to reconstruct RGBs from MSIs. And these two generators are formed into two loops ((2.2),(2.3)) by connecting one generator's output with the other generator's input.

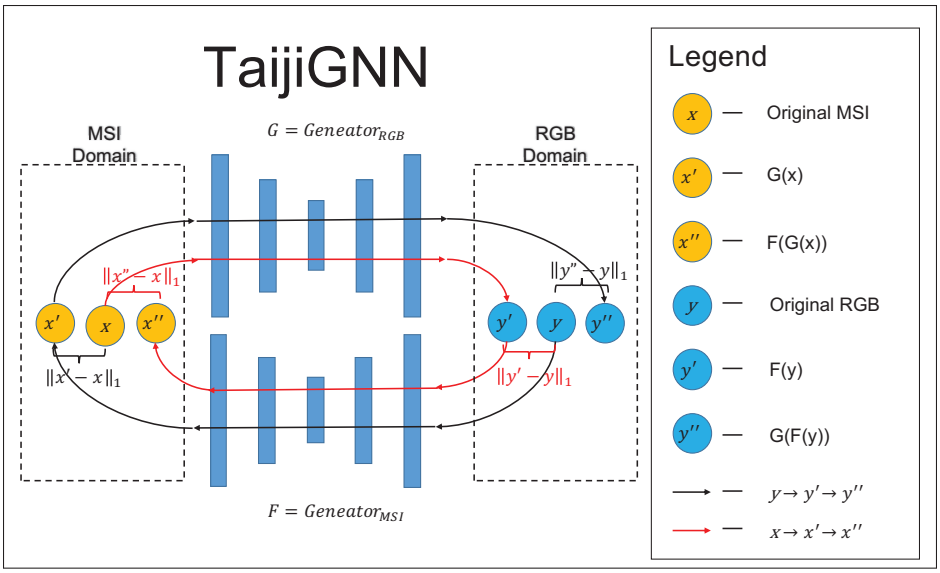

Figure 2.6 The detailed implementation of TaijiGNN

$$\underset{(Original)}{RGB} \overset{G_{MSI}}{\rightarrow} \underset{(Generated)}{MSI'} \overset{G_{RGB}}{\rightarrow} \underset{(CycleGenerated)}{RGB''} \tag{2.2}$$

$$\underset{(Original)}{MSI} \overset{G_{RGB}}{\rightarrow} \underset{(Generated)}{RGB'} \overset{G_{MSI}}{\rightarrow} \underset{(CycleGenerated)}{MSI''} \tag{2.3}$$

Moreover, since the paired ground truth RGBs and MSIs are available in the current scene, unlike CycleGAN, TaijiGNN uses supervised learning replacing GAN learning during training generators, G_{MSI} and G_{RGB}. Since one of the most significant shortages of GAN is their generated images are blurry, TaijiGNN avoids using GAN, which can enhance the clarity of the produced images.

When the training starts, the original RGBs and MSIs are input into the two loops, and the two training loops are activated simultaneously. One loop's input is the other loop's output, and at the same time, one loop's output is the other loop's input. Each generator is updated by three loss calculations. One is from its own output compared with the ground truth image. Another is from its cycled output compared with the ground truth. These two loss calculations are from

the same loop. Another is from the other loop. During the training process, the two generators complement each other to reach a dynamic balance.

Moreover, the TaijiGNN's training process coincides with the ancient Chinese philosophy "Taiji." In Taiji, "Yang" or "Positive" polarity and the "Yin" or "Negative" polarity consist of everything in the universe. Moreover, Yang comprises Yin and can be converted to Yin, and Yin contains Yang and can be converted to Yang under some circumstances. Sometimes, the two polarities complement each other, and sometimes they oppose each other. The final state of the two-pole system achieves a dynamic balance (Wikipedia, 2020c). I think Taiji's spirits and working mechanism are similar to our cycle consistent generative neural network, so I name the neural network "TaijiGNN."

The core idea of TaijiGNN is it translates the under-constrained spectrum conversion problem from comparing the two different domain images into comparing the same domain images. In this way, the two biggest spectrum translation challenges can be solved naturally.

CHAPTER 3

A HYBRID GPU-FPGA BASED DESIGN METHODOLOGY FOR ENHANCING MACHINE LEARNING APPLICATIONS PERFORMANCE

Xu Liu[1] , Hibat-Allah Ounifi[1] , Abdelouahed Gherbi[1] , Wubin Li[2] , Mohamed Cheriet[1]

[1] Université du Québec (École de technologie supérieure),
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Ericsson Canada,
8275 Trans Canada Route, Saint-Laurent, Québec, Canada H4S 0B6

Article published on the journal « Journal of Ambient Intelligence and Humanized Computing »
in June 2019,
DOI: 10.1007/s12652-019-01357-4

3.1 Abstract

The High-density computing requirements of Machine Learning (ML) is a challenging performance bottleneck. Limited by the sequential instruction execution system, traditional general purpose processors (GPPs) are not suitable for efficient ML. In this work, we present an ML system design methodology based on GPU and FPGA to tackle this problem. The core idea of our proposal is when designing an ML platform, we leverage the Graphics Processing Unit (GPU)'s high-density computing to perform model training and exploit Field Programmable Gate Array (FPGA)'s low-latency to perform model inferencing. In between, we define a model converter, which enable transforming the model used by the training module to one that is used by inferencing module. We evaluated our approach through two use cases. The first is a handwritten digit recognition with convolutional neural network (CNN) while the second use case is for predicting data center's Power Usage Effectiveness (PUE) with deep neural network (DNN) regression algorithm.

The experimental results indicate that our solution can take advantages of GPU and FPGA's parallel computing capacity to improve the efficiency of training and inferencing significantly.

Meanwhile, the solution preserves the accuracy and the Mean Square Error (MSE) while converting the models between the different frameworks.

3.2 Introduction

Recent massive adoption of machine learning applications, e.g., prediction of PM2.5 (Ganesh, Arulmozhivarman & Tatavarti, 2018) and mural deterioration detection (Huang, Feng, Fan, Guo & Sun, 2017), are commonly based on neural network algorithms such as Multilayer Perceptron(MLP) or CNN. These algorithms, in general, have to face substantial training data and consuming much time to achieve high accuracy, which is a challenge for GPPs (such as CPU) which have few computing cores and execute instructions in sequence and thus are worse at high-density computing tasks.

Meanwhile, people found that matrix operations almost dominate the ML algorithms. We can decompose the complex matrix operations into duplicated and straightforward atom operations such as additions and multiplications. Base on these facts and previous GPPs' shortages, people, try to find some other hardware devices which have much more parallel computing units. GPUs and FPGAs become the natural choice as they all have lots of computing units which can be re-programmed to work in parallel to increase the speed of machine learning.

A classical machine learning process includes two main phases, a training phase, and an inferencing phase. During the training phase, we pour tens of thousands of training data sets into the neural network, and the distance between the ground truth value and the prediction value will decrease continuously, and when the accuracy reach our goal, we stop the training and get the model. In the inferencing phase, we use the previous model to do prediction or estimation. When the training and inferencing are not on the same framework, for example, training on the Tensorflow and inferencing on Caffe, a model converter is necessary to convert the model from one framework to another.

Currently, there have been extensive studies on how to use only GPUs to accelerate the training phase (Raina, Madhavan & Ng, 2009; Bergstra, James et al., 2011; Sharp, 2008; Potluri, Fasih,

Vutukuru, Al Machot & Kyamakya, 2011) or how to use only FPGAs to accelerate the inferencing phase (Motamedi, Gysel, Akella & Ghiasi, 2016; Qiu *et al.*, 2016; Nagarajan, Holland, George, Slatton & Lam, 2011; Aydonat, O'Connell, Capalija, Ling & Chiu, 2017b), however, the investigation on how to combine the GPUs and FPGAs to improve the performance of the ML system, as well as what kind of hardware devices combination has the best performance remain mostly unexplored.

GPUs and FPGAs have different architectures, different characteristics, and performance. In our paper, we first analyzed the advantages and disadvantages when using GPUs and FPGAs individually to implement ML's various components, training, and inferencing. Then we gave out our design methodology of the ML system, and next, we performed two real ML cases, hand digital recognization, and prediction of datacenter's PUE based on our design methodology and tested and compared their performances. In the end, we made a discussion about the two experiments' results and gave out our viewpoints.

We organize the rest of our paper as follows: Section 3.3 presents related work about ML acceleration with different kinds of hardware devices. Section 3.4 elaborates the details of our hybrid design methodology of the ML system. Section 3.5 and Section 3.6 describe the implementations and performances of a CNN digital handwriting classification and a DNN PUE prediction based on our design methodology individually. Section 3.7 discusses the two cases' differences and problems we met and gives some explanations. In Section 3.8, we conclude our work and briefly discuss our future research plan.

3.3 Related work

People have studied ML acceleration with GPU extensively. For example, (Steinkraus, Buck & Y. Simard, 2005) implemented a full connected two layers neural network on ATI Radeon X800 graphic card and achieved 3X
speedup in training and testing. However, they published the paper in 2005; the primary method of using pixel shaders for ML computation is outdated. Similarly, (Raina *et al.*, 2009) used

GPU to accelerate the two unsupervised learning algorithms, including deep belief networks (DBNs) and sparse coding. They took advantages of GPUs' global memory to save the data and parameters from reducing the transfer time between the host machine and the GPU, resulting in performance improvement with 70 times faster than a dual-core CPU when implementing the DBNs algorithm. Their result demonstrated the potential of acceleration using GPUs, which is also confirmed by (Potluri *et al.*, 2011) in their work where Potluri et al. have used GeForce 9500 GT with 256MB memory graphic card to speed up GPU-based Universal Machine - CNN (UM-CNN).

On the adoption of FPGAs to accelerate ML algorithms, (Motamedi *et al.*, 2016) presented an accelerator for deep CNNs. Their accelerator can exploit the available parallelism resources to minimize the execution time, achieving a $1.9X$ speedup comparing with the state-of-the-art deep CNN accelerator. (Aydonat *et al.*, 2017b) proposed a new architecture designed with OpenCL. They tried to increase data reusing to reduce external memory bandwidth consumption. They also used the Winograd transform to improve the FPGAs' performance. They managed to speed up executing the AlexNet CNN benchmark on Intel's Arria 10 FPGA, $10X$ faster than the state-of-the-art on FPGAs and the power efficiency is similar to the best implementation of AlexNet on TitanX GPU. (Nagarajan *et al.*, 2011) has proposed a method to implement a multi-dimensional PDF estimation algorithm which used Gaussian kernels on the FPGA. They used ActiveHDL to develop their platform. It is a hardware description language (HDL) and is not so popular and not so easy to use. They have got a $20X$ speedup over a 3.2 GHz CPU processor.

To the best of our knowledge, few works combined FPGA and GPU to improve the performance of ML systems in the past. The first work we found is a FPGA-GPU architecture for kernel SVM pedestrian detection by (Bauer, Köhler, Doll & Brunsmann, 2010). They used GPU for model training and inferencing and used FPGA for feature extraction. In the second work, (Zhu, Liu, Wang & Xie, 2016) have implemented a novel parallel framework for neural networks with GPU and FPGA. In their work, the neural network processing was decomposed into layers and scheduled either on the GPU or FPGA accelerators. Additionally, in a white paper (Advanced

Micro Devices, Inc. and Xilinx, Inc., 2017), without giving detailed information, the authors only gave a hypothesis that the combination of CPU, GPU and FPGA would have the best performance, but did not give any verification.

In summary, few studies have combined GPUs and FPGAs together to implement ML systems, not to mention how to convert models from one framework to another.

3.4 Design methodologies

3.4.1 The whole architecture and workflow

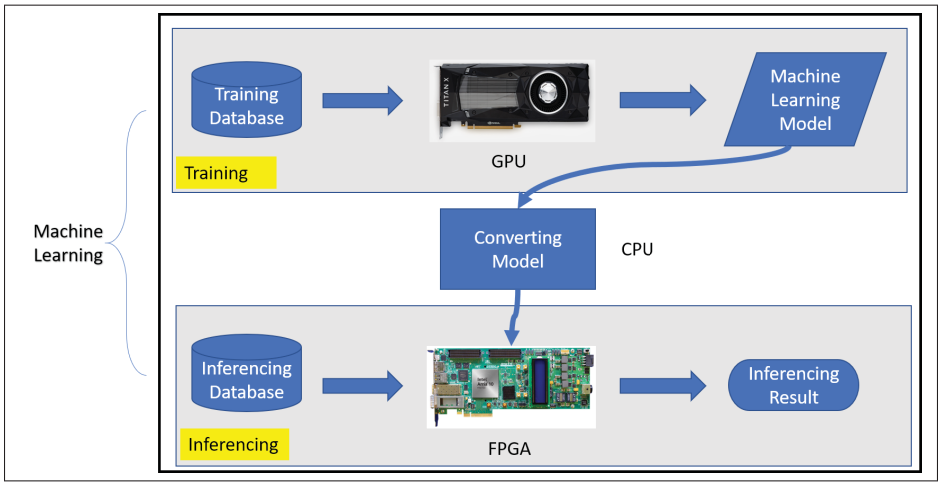

Figure 3.1 The architecture of heterogeneous machine learning system
Taken from Liu et al. (2018, p. 104)

A standard ML system usually has two main modules; one is for training model, the other is for inferencing model. Model is the belt between the training module and the inferencing module.

Fig. 3.1 presents the whole architecture of our ML system, which contains three modules, i.e., training module, inferencing module, and converting-model module. We implemented the training module on the GPU and built the inferencing module on the FPGA. As the FPGA inferencing framework is different from the GPU training framework, a model converting component is needed.

The entire flow is as follows. The training module loads training data from the database first and then uses ML to train the model. When the model is ready, the system converts the model from the training framework to the inferencing framework. In the end, the inferencing module loads the model file and the inferencing data and does classification or prediction.

3.4.2 The GPU training module

The goal of the training phase is to train a model to obtain the maximum test accuracy within minimum time. Based on two reasons, we select the GPU to implement the training module. One reason is, comparing with FPGAs, GPUs often own lower price/performance ratio in model training. So most research projects select GPUs to do training (Steinkraus *et al.*, 2005; Raina *et al.*, 2009; Potluri *et al.*, 2011), and few projects choose FPGAs to train their model (Zhao *et al.*, 2016). The other reason is the training module usually has complex architectures (forward and backward propagation, gradient descent, and so on) and its goal is to get the model, once get the model, the training module is useless, so people hope to build the training module quickly. On this point, GPUs are much easier to program than FPGAs. As GPUs have similar instruction system as CPUs, the programs run on CPUs can be easily transplanted to GPUs. Furthermore, Nvidia corporation has created CUDA (Compute Unified Device Architecture), which is a GPU computing program standard based on the C programming language. Moreover, many companies have already developed a series of frameworks which support CUDA standard and hide CUDA implementation details, allowing users to focus on the design of ML algorithms. TensorFlow is almost the best one among those frameworks, which is developed by Google Brain (Google Inc., 2018).

Fig. 3.2 is our training module development flow. Firstly we use Tensorflow to design our training algorithm, and then the Tensorlfow will translate the python code into CUDA code, and depart the code into two parts, and put one on the GPU to run, put the other on the CPU to execute. After finishing GPU computing, the CPU will collect the results from the GPU and combine the results.

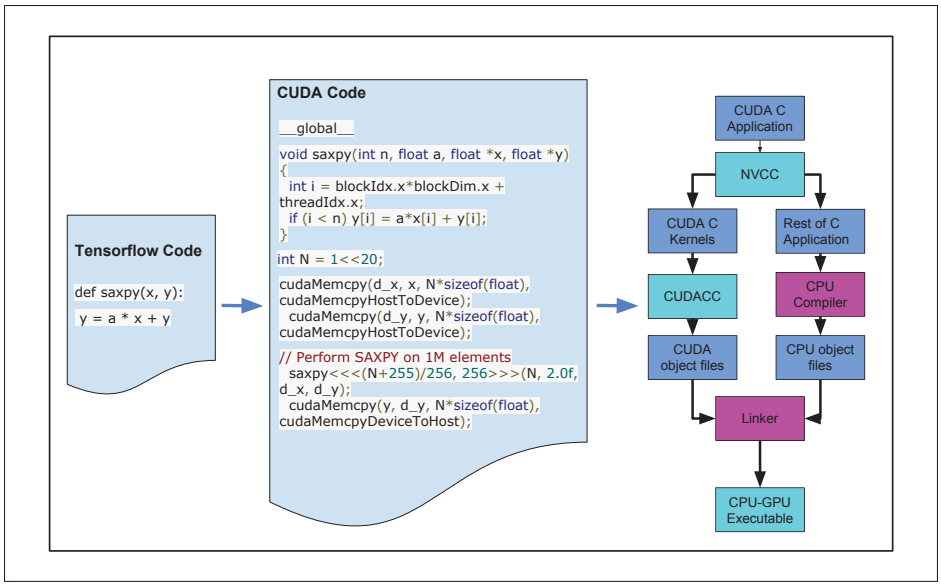

Figure 3.2 The flow of the GPU training development
Taken from Timothy et al. (2013, Website)

3.4.3 The FPGA inferencing module

The goal of the inferencing phase is to do inferencing with minimum latency using the model generated from the training phase. Unlike training, we often do inferencing multiple times. Any small latency of each cycle can accumulate to a considerable amount. So it is valuable to reduce the inferencing latency.

FPGAs are composed of logical elements (LEs). We just programmed these LEs into different hardware electrical circuits to meet our requirements. FPGAs' performances are near Application-specific integrated circuits (ASICs), and they can be reconfigured many times, so their prices are usually much higher than ASICs, CPUs, and GPUs. In most FPGA designs, there is no fetching and decoding instruction system, which make FPGAs are much faster than GPUs and CPUs. However, this advantage also becomes the FPGAs'disadvantage. As the FPGAs do not have traditional instruction system, when we have a new function requirement, we can not use the general software programming language such as C to describe it. We have to use some HDLs such as Verilog or VHDL (VHSIC very high-speed hardware description language) to re-design

the whole hardware electrical circuit. The HDLs are similar to the assembly languages, and even a small function needs many HDL sentences to describe. If a system is as complex as the DNN's training module, it is hard to use HDLs to implement it, which restricts the scales of FPGA designs most.

Now, we can compensate the difficulties of FPGA design by using high-level programming languages such as OpenCL (Open Computing Language) which do not require too much electrical circuits knowledge (Aydonat *et al.*, 2017b; Zhao *et al.*, 2016; Bettoni, Urgese, Kobayashi, Macii & Acquaviva, 2017; Li, Liu, Xu, Yu & Ren, 2018). OpenCL is a framework for developing programs which can run on different heterogeneous platforms such as GPU + CPU or FPGA + CPU.

Fig. 3.3 shows its development flow. We can divide the development into two phases: the host program and the kernels. When designing the kernels, we should consider how to take the full parallel computing capacity of the FPGA. The kernels will be synthesized into hardware logic circuits and uploaded to the FPGA development board. The host program, which is similar to the traditional C program that runs on the CPU, is in charge of allocating parallel computing jobs on the FPGA development board and collecting the computing results from the FPGA.

3.4.4 Model converter

Training and inferencing are usually put on different platforms, as the training process is high-density computing and time-consuming, it needs a large number of computing resources which is not affordable for most inferencing platforms. Moreover, the training is usually a one-off process in a fixed time. On the contrary, the inferencing happened often. So it is a smart and efficient way to use different platforms to implement the training and inferencing phases.

However, generally, different platforms adopt different frameworks and model definitions. We can not use the model generated by one framework on another directly. A model converter which can convert a model from one framework to another is needed. Although there are already some tools such as MMdnn (Chen, Cheng et al., 2019) for converting the model file among the main

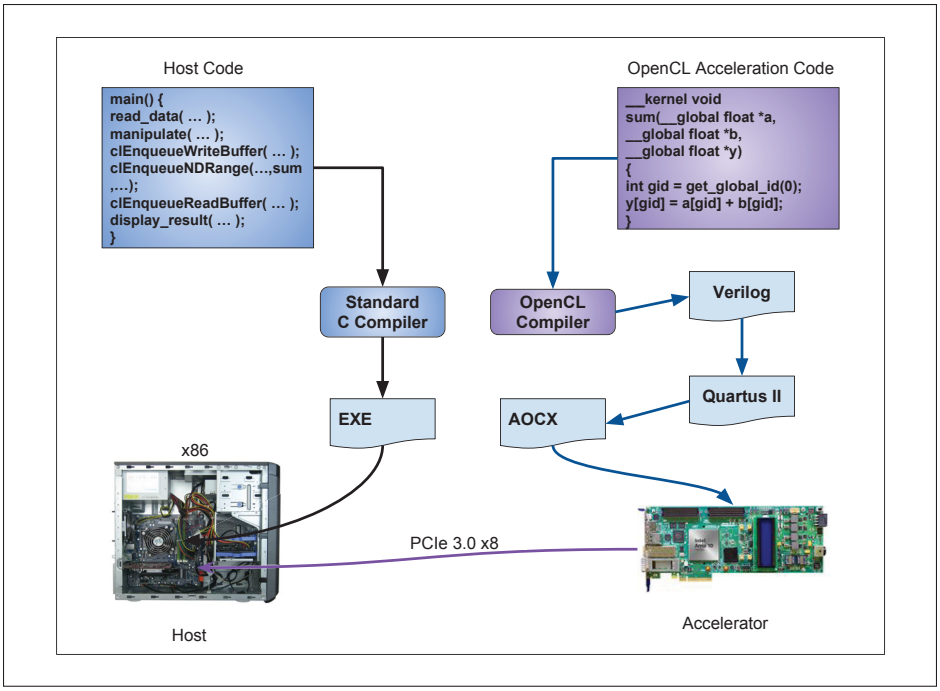

Figure 3.3 The flow of the FPGA inferencing development
Taken from Intel (2018, Website)

frameworks (Tensorflow, Caffee, PyTorch and so on), it is useless for personal custom-made cases, including our case.

Suppose we should turn a model from the source framework to the target framework. For converting a model, here is a general process:

1. Understand and capture the date structure of the source model completely;
2. Understand and capture the model file definition of the target framework fully;
3. Design a mapping function from the source model parameters and weights to the target model.

A model data structure or a model file of machine learning contains parameters such as weights and biases generated during the training process, it is the core and goal of machine learning, and we save it after training and load it before inferencing.

It is not accessible to understand a model data structure or file well, as there are many items, such as the number of layers, the size of each layer, the size of each filter, the activation functions used between layers, the order of matrix dimensions, and the flattening ways. In particular, if the model definition is not open source, we have to guess the order of matrix dimensions, which is almost the hardest part, as the matrices involved usually have 4−dimensions, leading to $4 \times 3 \times 2 \times 1 = 24$ possible flattening ways. However, only one way is correct.

When we design the model mapping function, we should obey all of the above definitions in the target model, which is the secret to guarantee the correctness of model converting.

Fig. 3.4 is a simple model converting example. There are two frameworks, 1 and 2, which have the same machine learning architectures and the numbers of parameters. However, their model's data structures and flattening ways are different. W_1 in model 1 is stored as $\begin{bmatrix} 1 & 1 \\ 0 & 0 \end{bmatrix}$ and flattened as $[1, 1, 0, 0]$ while W_2 in model 2 is stored as $\begin{bmatrix} 0.1 & 0.1 \\ 0 & 0 \end{bmatrix}$ and is flattened as $[0.1, 0, 0.1, 0]$. After understanding the two model's data structures and flattening ways well, it is easy to convert the model from framework 1 to framework 2.

In our case, as we implemented the training module on the Tensorflow framework, and implemented the inferencing module on the OpenCL-FPGA framework, we design a particular model mapping function which can map weights and bias from the Tensorflow framework to the FPGA framework.

3.4.5 Summary

After analyzing and comparing different hardware device combinations, we found the best solution to implement a machine learning system that is to use GPU to implement the training module and the FPGA to implement the inferencing module and add a model converter in charge of converting the model from one framework to another.

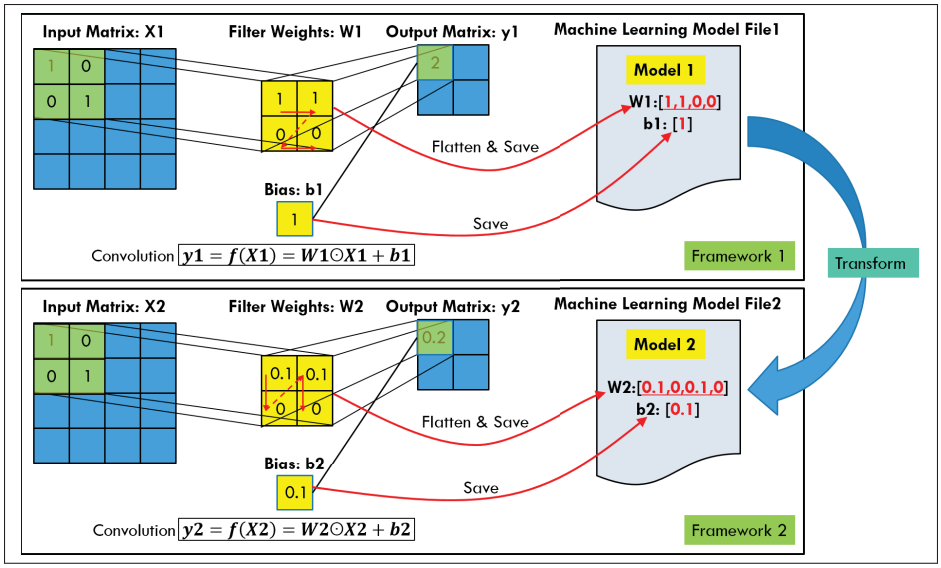

Figure 3.4 Schematic of model converting
Taken from Liu et al. (2018, p. 105)

To verify this design methodology, we implemented two ML use cases and tested their performance. The first is a CNN for digital hand-writing recognition, the other is a DNN regression for estimating the PUE of the data center.

3.5 A case study on digital handwriting recognition with CNN classification

3.5.1 Experiment environment

The following two cases use the same experiment environment.

3.5.1.1 Hardware environment

Our devices' list is shown in Table 3.1. We use the motherboard to host every hardware device together. The Titan XP graphics card communicates with the host board through PCIe gen 3 x 16 (with the bandwidth of 15760 MB/s). The Arria 10 development board transfers data to the host board by PCIe gen 3 x 8 (with the bandwidth of 7880 MB/s). When the amount of training

data is huge, GPU cannot load the whole data in one-time and has to break the data into many small batches, which will lead to transferring data frequently between the host and the device.

On the contrary, the data for inferencing is usually small, that is no need to transfer data frequently. From this perspective, the bandwidth becomes a performance bottleneck of the system. Therefore, this becomes another reason to use the GPU to do the training and the FPGA to do the inferencing.

Table 3.1 Hardware devices list
Taken from Liu et al. (2018, p. 106)

Device	Type	Number
CPU	Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz	1
Memory	Samsung DDR4 8g	4
Solid State Disk Drive	INTEL SSDSC2BB48 480g	1
Mechanical Hard Disk Drive	WDC WD40EFRX-68N 4TB	1
GPU	NVIDIA TITAN Xp	1
FPGA	Intel Arria 10 GX FPGA Development Kit	1

3.5.1.2 Software environment

Table 3.2 is our software environment. The operating system is Ubuntu 16.04. We use the Tensorflow to implement the GPU training module and the OpenCL to design the FPGA inferencing module.

Table 3.2 Software environment
Taken from Liu et al. (2018, p. 106)

Software	Version
Ubuntu	16.04.3 LTS
Python	3.5.2
Tensorflow	1.4.0
Tensorflow-GPU	1.4.0
CUDA	8.0
Intel(R) FPGA SDK for OpenCL	17.1.0 Build 240

3.5.2 The algorithm of CNN case

Our goal is to verify the performance of the heterogeneous machine learning system. We select LeNet-5 and MNIST (LeCun, Yann and Cortes, Corinna and J.C. Burges, Christopher, 2018) as our algorithm and data set, which are not too complicated but enough to show the effects of hardware acceleration.

The MNIST is a training dataset for digital handwriting recognition. Each example is a pixel value matrix whose size is 28 x 28, and each pixel value's range is from 0 to 255. In our case, for easy processing, we divide the pixel value by 255 (Fig. 3.5). MNIST dataset totally includes 55, 000 training examples, 5, 000 validation examples, and 10, 000 test examples.

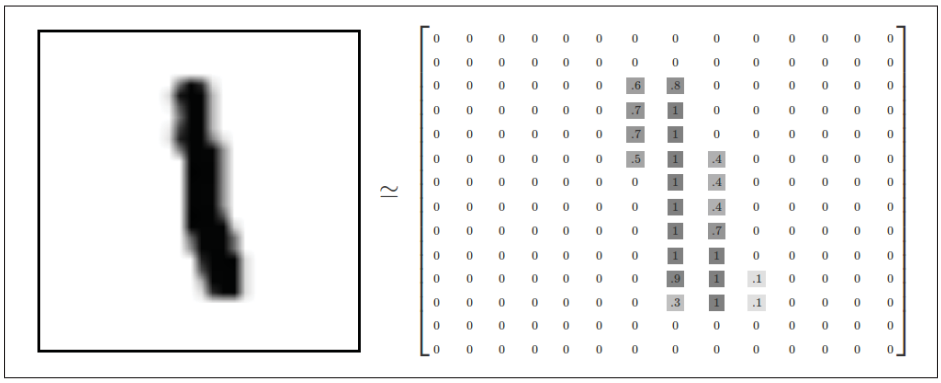

Figure 3.5 The pixel value matrix
Taken from Tensorflow (2018, Website)

The LeNet-5 is designed by Yann LeCun for handwritten and machine-printed character recognition (LeCun, Bottou, Bengio & Haffner, 1998), belongs to CNNs. It has almost all classical CNN structures (convolution layers, pooling layers, and full connection layers, and so on) and has high accuracy in digital handwriting recognition. Most importantly, it contains a lot of high-density computing jobs which are suitable for testing the acceleration effects of parallel designs.

Fig. 3.6 is our tailor-made version of LeNet-5 for this experiment. It has 7 layers in total, including 3 convolution layers, 2 sub-sampling layers, and 2 full connection layers. These layers are orderly arranged, as illustrated in Fig. 3.6.

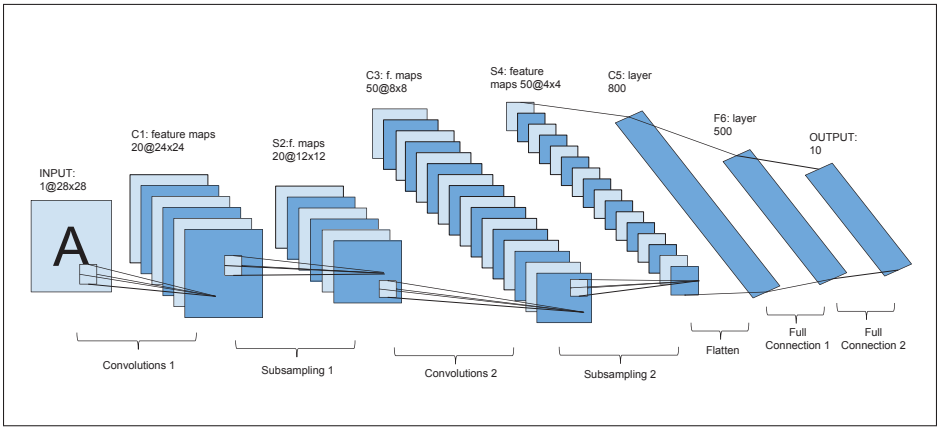

Figure 3.6 The architecture of LeNet-5 improved for experiment
Taken from Liu et al. (2018, p. 106)

3.5.3 The training module of CNN case

3.5.3.1 The implementation

Fig. 3.7(a) is our training phase control flow diagram. It includes two main phases, i.e., forward propagation and backward propagation. The difference of value calculated by the forward propagation and label value is passed into the backward propagation to calculate the weights and biases of each layer. After several computing loops, the difference converges within a permissible range, the training process terminates. We store weights and bias in a model file.

We use NVIDIA Titan Xp graphics card to accelerate the training process. The GPU has 3,840 CUDA cores which can be programmed to do parallel computing. Its floating computing performance can reach 12 TFLOPS.

To do the NVIDIA graphics card computing development, it should use the CUDA programming language mentioned previously. That will mean we should build the acceleration kernel from

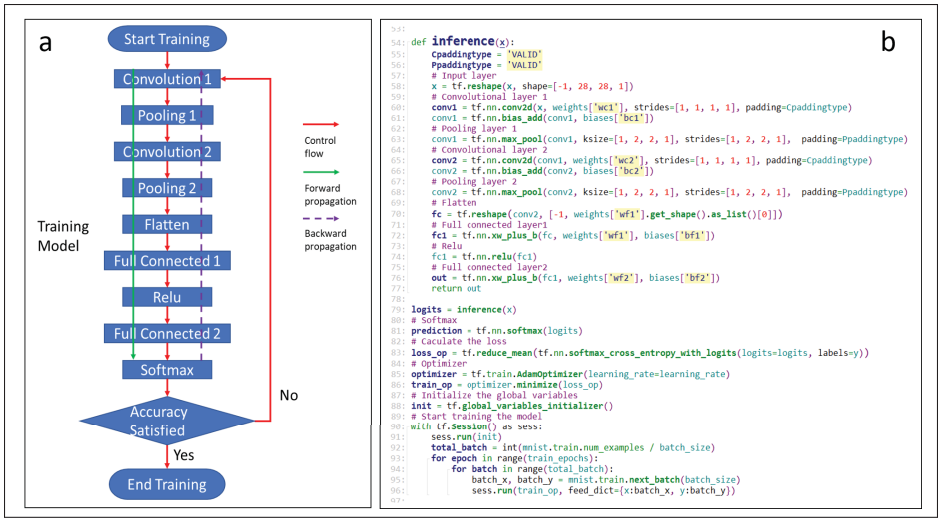

Figure 3.7 (a) The control flow of Training (b) The TensorFlow implementation of the training module Taken from Liu et al. (2018, p. 106)

scratch. Thanks to the TensorFlow, which packages the CUDA libraries, now we only need to focus on designing the architecture. The Fig. 3.7(b) is part of our TensorFlow implementation code.

3.5.3.2 The experiment results

In this experiment, we aim to find a better device for training model by comparing the training speed of CPU and GPU.

For achieving this goal, we designed two use cases. One only has a CPU-E5-1620, the other has a CPU-E5-1620 and a GPU-TitanXp. We chose the same 55,000 training examples and measured their training time, respectively. We did six times the same tests and calculated their average times.

The results are in Table 3.3. We can conclude that the average speed of TitanXp is about 8.8x faster than the average speed of CPU E5-1620 with the same accuracy. This result is similar to (Tobias Kind, 2018)'work whose GPU speed is about 9x faster than CPU.

Table 3.3 The results of CNN training time
Taken from Liu et al. (2018, p. 106)

Experiments	CPU-E5-1620		GPU-TitanXp		Accelerate Times
	Training Time(*s*)	Accuracy (*%*)	Training Time (*s*)	Accuracy (*%*)	Acceleration (*GPU/CPU*)
1	448.60	98.70	50.61	98.80	8.86
2	447.79	98.88	51.17	98.58	8.75
3	448.35	98.90	50.73	98.80	8.84
4	448.62	98.94	50.46	98.88	8.89
5	447.62	98.59	50.94	98.82	8.79
6	447.88	98.94	50.78	98.79	8.82
Average	448.10	98.80	50.80	98.80	8.80

3.5.4 The inferencing module of CNN case

3.5.4.1 The implementation

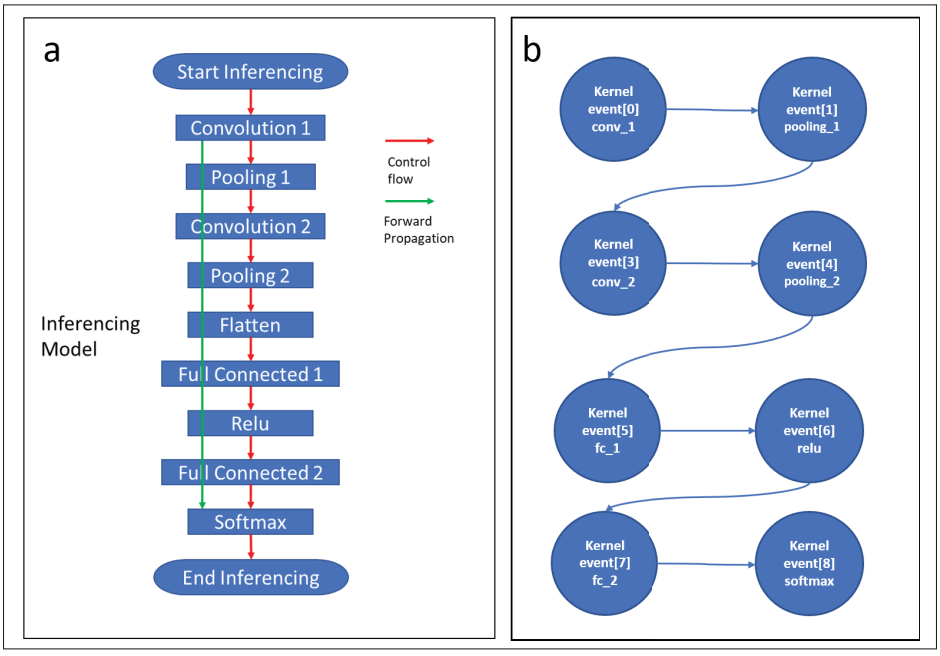

Figure 3.8 (a) The control flow of inferencing (b) The kernels of OpenCL. implementation
Taken from Liu et al. (2018, p. 106)

Fig. 3.8(a) is our inferencing phases control flow. It only has forward propagation. It loads the model generated from the training module and then does inferencing only with the forward propagation algorithm.

We use Intel Arria 10 FPGA development board to implement the inferencing module. Arria 10 is made by Intel with the 20nm process. It has high performance and low power consumption. Although its 1.5 TFLOPS is still slower than TITAN Xp 12 TFLOPS, its power consumption below 100 watts is much better than TITAN Xp 250 watts.

We use OpenCL for FPGA development. Its development includes two parts: a host program and kernels. The host program runs on the CPU, and the kernels run on the FPGA. Fig. 3.8(b) shows our kernel events implemented with OpenCL. It has 8 kernel events which map with the control flow's 8 functions. These logical kernel events will be programmed on the FPGA and executed one by one to implement the inferencing function.

3.5.4.2 The experiment results

In this experiment, we use the CPU, GPU, and FPGA to execute the same inferencing algorithm with the same model and compare their efficiencies with each other.

First, we measure the time of inferencing $10,000$ images on the CPU, GPU, and FPGA devices, respectively. Then we execute the measurement 6 times and calculate the average time of inferencing $10,000$ images on different hardware devices.

The results of the experiments are presented in Table 3.4. From the table, we can see that the average speed of Arria 10 is about 10.9 times faster than the average rate of CPU E5-1620 and is about 7.1 times faster than the GPU TitanXp.

Table 3.4 Results of 10,000 image inferencing time

Experiments	CPU-E5-1620	GPU-TitanXp		FPGA-Arria10		
	Inferencing Time (us)	Inferencing Time (us)	Acceleration (GPU/CPU)	Inferencing Time (us)	Acceleration (FPGA/CPU)	Acceleration (FPGA/GPU)
1	14.881587	9.713557	1.5320	1.38002	10.8	7.0
2	14.995880	9.202130	1.6296	1.34051	11.2	6.9
3	14.685811	9.419498	1.5591	1.36456	10.8	6.9
4	14.785795	9.762074	1.5146	1.34304	11.0	7.3
5	15.064704	9.854970	1.5286	1.32339	11.4	7.4
6	14.307688	9.440861	1.5155	1.37961	10.4	6.8
Average	14.786910	9.565520	1.5459	1.35520	10.9	7.1

3.5.5 The model converter of CNN case

3.5.5.1 The experiment results

For verifying our model converter works well, we conduct two experiments on the same 10K MNIST test examples respectively. In Experiment_1, we only test FPGA inferencing accuracy with the original model while in Experiment_2, and we change CNN's configuration, retrain the model by TensorFlow with GPU, convert the model to the FPGA and then test the inferencing accuracy on FPGA and Tensorflow respectively.

Table 3.5 presents the statistics of accuracy collected from the two experiments. The Experiment_-2's accuracy is better than Experiment_1's. In Experiment_2, the FPGA has preserved the same precision as the TensorFlow, which proved our model convert successfully.

Table 3.5 Converting model experiments of CNN

Experiment_1	Accuracy(%)	Experiment_2	Accuracy(%)
N/A	N/A	TensorFlow	99.13
FPGA	99.05	FPGA	99.13

3.6 A case study on data center PUE with DNN regression

3.6.1 The overview of the use case

To fully evaluate our hybrid design methodology of the ML system, we select another algorithm to implement and test its performance on CPU, GPU, and FPGA. This algorithm belongs to one of DNN regressions, and we use it for estimating the PUE of Data Center (DC).

A DC is a physical space that groups together IT systems (servers, storage, and so on.), mechanical systems (Computer Room Air Conditioner (CRACs), Chillers, and so on), and electrical systems (Uninterruptible Power Supply (UPS), Power Distribution Unit (PDU), transformers, and so on), for storing, processing and protecting data. The energy consumption takes the main part of operating a DC and can reach up to 75% of operating costs, which is one of the major reasons why industrial and environmental organizations have focused on improving energy performance while ensuring continuity of services. For this reason, many energy-related metrics are defined, such as PUE. The equation (3.1) is the definition of the PUE (Datacenterknowledge, 2019). The IT equipment power includes all the actual load of IT equipment such as workstations, servers, storage, switches, printers, and other service delivery equipment (Datacenterknowledge, 2019).

$$PUE = \frac{TotalDatacenterPower}{ITEquipmentPower} \tag{3.1}$$

The PUE stands for how energy is efficiently used to keep the DCs running without service interruption. It is used to evaluate over a year the total amount of energy consumed by the DC, compared to the amount of energy necessary for the operation of the IT equipment. The closer the result is to 1.0, the less power the non-IT equipment consumes, and the more it is considered "eco-responsible".

Moreover, the interactions of DC systems are complicated. According to (Ounifi *et al.*, 2018) 's work, the DC systems (IT, electrical and mechanical systems) interactions and the different feedback loops make it difficult to estimate and predict the DCs' energy efficiency accurately.

To capture such complexities, we try to find an estimation model to calculate the PUE metric values with the help of DNN.

DNN borrowed the concept of the deep neural network of the brain and will mainly analyze and process the input data through a succession of several neurons that take the input signals from the previous neurons. DNNs are good at modeling non-linearity and have characteristics such as the ability to model real-time operation and fault tolerance.

In this case study, we will try to use our machine learning design methodologies to implement a DNN regression system which can train a model for estimating the PUE of a data center.

3.6.2 The dataset and the DNN regression algorithm of DNN case

3.6.2.1 The dataset

The dataset we used is from the "ITEA3 RISE SICS Data Center" located in Sweden. It has 2881 sets. Each set is collected from the DC every 60 seconds from 9 a.m. to 9 p.m. containing 415 kinds of DC features such as fan speed, input DC's power, and average cold Aisle temperature (Ounifi *et al.*, 2018). Table 3.6 shows part of the 415 features. Besides, the dataset also has a time series column and a ground truth PUE column. So our working dataset is a 2881 x 417 matrix. We apply cross-validation by dividing the 2881 data sets into two parts: 2656 for training sets, and 225 for test sets.

Table 3.6 A part of selected DC features of the ITEA3 RISE Sics DC

DC Features	Units
Indoor/outdoor temperature	$°C$
Input Data center power	Mw
Whole Data center humidity	%
Energy consumption/ rack	Mw
Workload (electrical)/server	Mw
Workload (CPU Usage /server	Mgbit
Power consumption after the PDU	Mw
Average cold Aisle temperature	$°C$
Fan speed	RPM
Fan power	Kw
CRAC Fan power	Kw
Power Used by chilled liquid	Kw
Chilled water entering temperature	$°C$
CRAC energy consumption	KVA
Total Rack IT load	KVA
Hot Aisle temperature	$°C$
hline Outside air dry bulb temperature	$°C$

3.6.2.2 The architecture of DNN regression algorithm

According to the structure of Section 3.6.2.1's data sets, we design a tailor-made version of DNN regression which is composed of 1 input layer, 5 hidden layers and 1 output layer. The details architecture of the 5 hidden layers is as follows:

- Input layer: 415 neurons.
- Full connected layer 1: 512 neurons.
- Activation function: Relu.
- Full connected layer 2: 1024 neurons.
- Activation function: Relu.

- Full connected layer 3: 1024 neurons.
- Activation function: Relu.
- Full connected layer 4: 1024 neurons.
- Activation function: Relu.
- Full connected layer 5: 512 neurons.
- Activation function: Relu.
- Output layer: 1 neuron.

And all these have been shown on the Fig. 3.9.

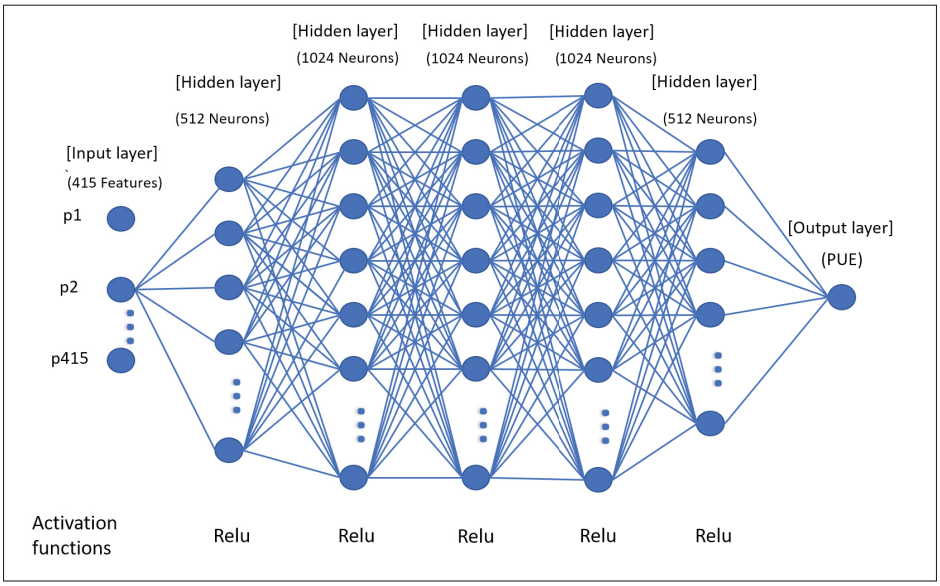

Figure 3.9 DNN proposed model

3.6.3 The training module of DNN case

3.6.3.1 The implementation

Fig. 3.10(a) is our training control flow which is similar to Fig. 3.7(a). It includes two main phases, i.e., forward propagation and backward propagation. The difference between CNN case and DNN case is the DNN case has no convolutional layers, and pooling layers and the cost

function is Mean Square Error (MSE) which defined in equation (3.2) instead of Mean Cross Entropy (MCE). After several computing loops, the MSE will converge within a permissible range, the training process terminates. We will store the final weights and bias in a particular model file.

$$MSE = \frac{1}{n}\Sigma_{i=1}^{n}(y_i - y_i')^2 \tag{3.2}$$

where y_i is the real output, y_i' is the calculated output, and n is the number of the input data. We still use Tensorflow to implement this training part. Fig. 3.7(b) is part of our TensorFlow code.

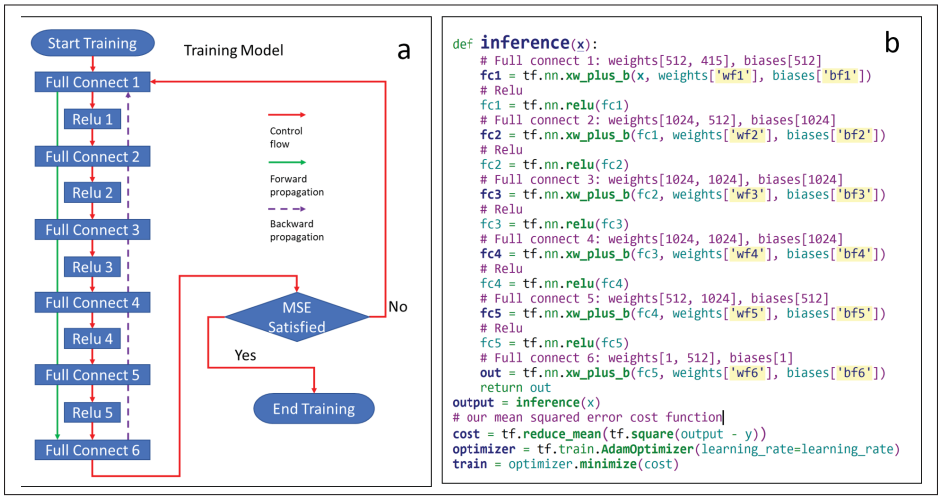

Figure 3.10 (a) The control flow of DNN training (b) The TensorFlow implementation of the DNN training module

3.6.3.2 The experiment results

During this experiment, we run the same DNN regression training program with the same training dataset on GPU-TitanXp and CPU-E5-1620 respectively and then measure their MSEs and execution time.

Fig. 3.11 is the result that we use the model trained by GPU and the model trained by the CPU to do PUE inferencing on the same test set. They look almost the same. They have the same

MSE: 0.000029. However, their execution times are different. Table 3.7 shows that the GPU's training speed is about 12.4x faster than the CPU's.

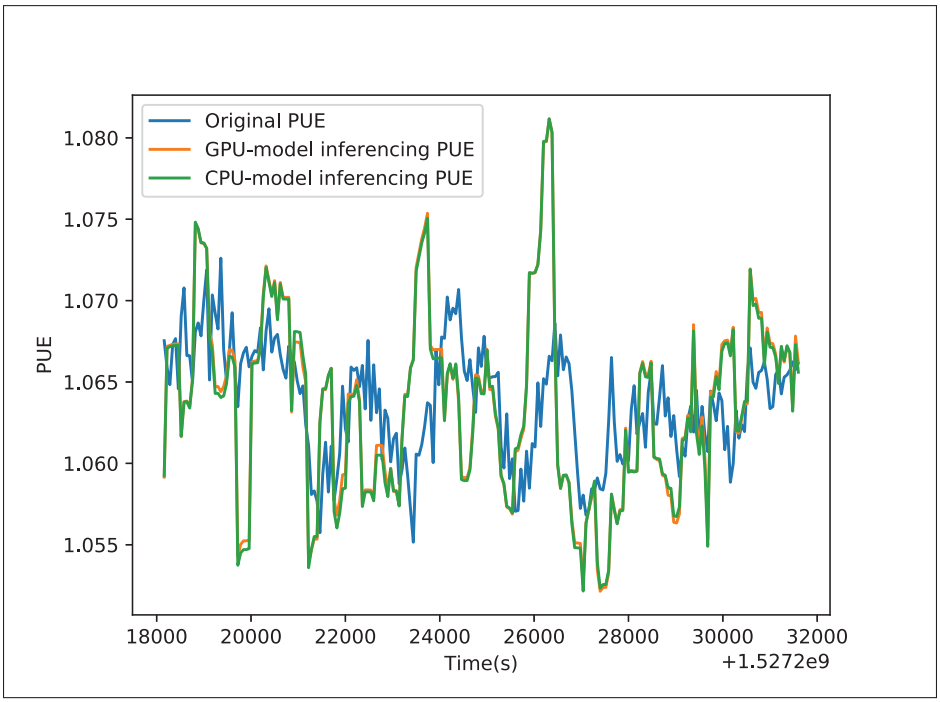

Figure 3.11 The comparison of GPU-model and CPU-model inferencing results

Table 3.7 The DNN training time

Experiments	CPU-E5-1620		GPU-TitanXp		Accelerate Times
	Training Time(*s*)	MSE (*%*)	Training Time (*s*)	MSE (*%*)	Acceleration (*GPU/CPU*)
1	154.36	0.000029	12.14	0.000029	12.72
2	154.58	0.000029	12.38	0.000028	12.48
3	154.66	0.000029	12.52	0.000029	12.35
4	154.74	0.000029	12.40	0.000029	12.48
5	154.49	0.000029	12.68	0.000029	12.12
6	154.78	0.000029	12.73	0.000029	12.16
Average	154.60	0.000029	12.50	0.000029	12.40

3.6.4 The inferencing module of DNN case

3.6.4.1 The implementation

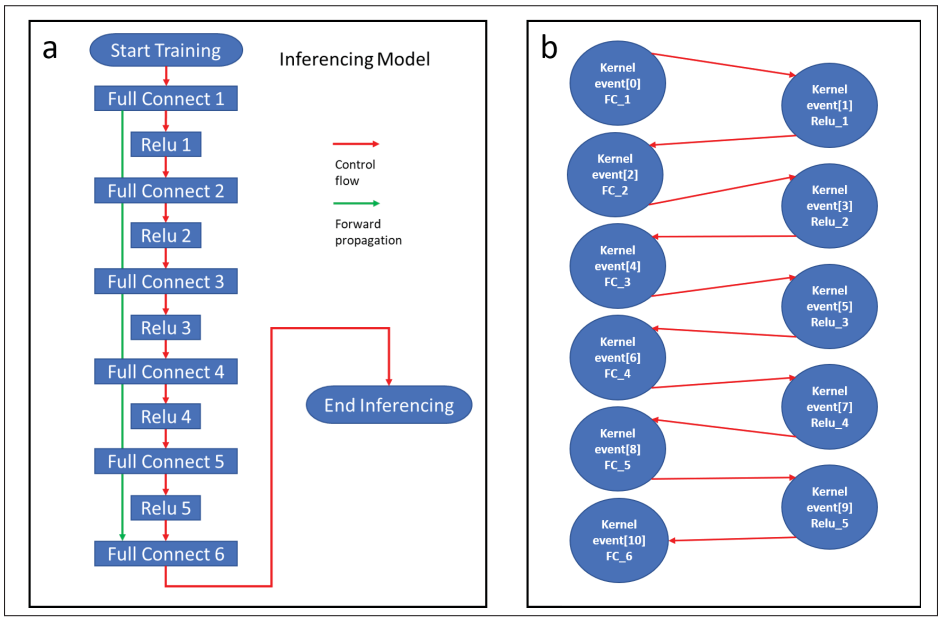

Figure 3.12 (a) The control flow of DNN inferencing (b) The kernels of OpenCL DNN implementation

Fig. 3.12(a) is our inferencing phases control flow, which only has two operations, full connecting and Relu. It loads the model generated from the training phase and then does inferencing only with the forward propagation algorithm.

We also use OpenCL for FPGA design. Like GPUs, the process of FPGA development has two parts: a host program and kernels. The host program which runs on the CPU is in charge of loading datasets and model, allocating computing jobs on the FPGA and collecting the results from the FPGA. The kernels run on the FPGA are the primary computing acceleration part.

Fig. 3.12(b) shows our kernel events implemented with OpenCL. It has 11 kernel events which map with the control flow's 11 functions. These logical kernel events will be programmed on the FPGA and executed one by one to implement the inferencing function.

3.6.4.2 The experiment results

Similar to the previous use case, we use the CPU, GPU, and FPGA to execute the same inferencing algorithm with the same model and compare their inferencing times with each other. First, we measure the time of inferencing 225 sets on the CPU, GPU, and FPGA devices, respectively. Then we execute the measurement 6 times and calculate the average time of inferencing 225 sets on different hardware devices.

The results of the experiments are presented in Table 3.8. From the table, we can see that the average speed of FPGA Arria-10 is about 13.6X times faster than the CPU E5-1620 and is about 3.7X times faster than the GPU TitanXp.

Table 3.8 Results of 225 sets inferencing time

Experiments	CPU-E5-1620	GPU-TitanXp		FPGA-Arria10		
	Inferencing Time (s)	Inferencing Time (s)	Acceleration (GPU/CPU)	Inferencing Time (s)	Acceleration (FPGA/CPU)	Acceleration (FPGA/GPU)
1	0.539186	0.160862	3.3519	0.0363316	14.8	4.4
2	0.488358	0.133327	3.6629	0.0374917	13.0	3.6
3	0.508489	0.119494	4.2554	0.0346079	14.7	3.5
4	0.500365	0.122232	4.0936	0.0382779	13.1	3.2
5	0.520802	0.129460	4.0229	0.0384238	13.6	3.4
6	0.451343	0.153649	2.9375	0.0362708	12.4	4.2
Average	0.501420	0.136500	3.6733	0.0369006	13.6	3.7

3.6.5 The model converter of DNN case

3.6.5.1 The experiment results

Since the framework of Tensorflow is different from the framework of our FPGA, we need to undertake the model converting. To verify that our model converter works correctly, we test the MSE of Tensorflow and the MSE of FPGA on the same 225 test data sets using the model generated by the Tensorflow and the model converted by our platform, respectively. Table 3.9 are the results. We have done 6 times training on the Tensorflow framework and got 6 models,

converted these models for the FPGA, and with these converted models we do inferencing on the FPGA, and all got the same MSE.

Table 3.9 Converting model experiments of DNN

Experiments	MSE of Tensorlfow	MSE of FPGA
1	0.000028	0.000028
2	0.000029	0.000029
3	0.000028	0.000028
4	0.000029	0.000029
5	0.000029	0.000029
6	0.000029	0.000029

Fig. 3.13 is one of our test case results. From this figure, we can see that the inferencing results of Tensorflow are almost the same as the results of FPGA, which proves our model converter working well.

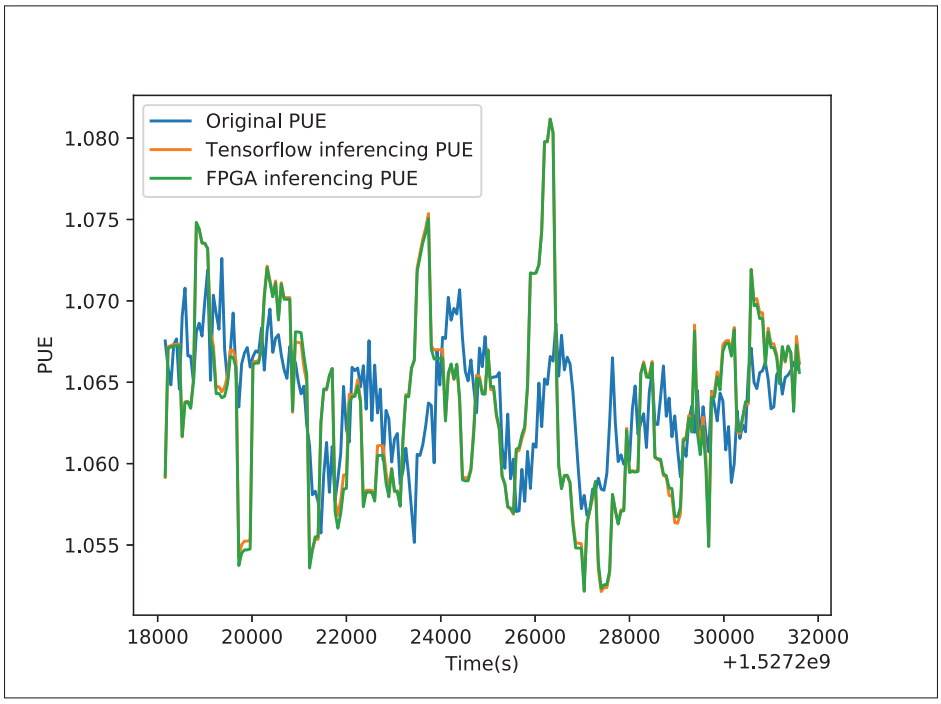

Figure 3.13 Results of converting model in inferencing PUE

3.7 Discussions

We intentionally selected two different area use cases to verify that our hybrid ML platform design methodology is general. Although the two algorithms are different, one is CNN, the other is normal DNN, with our solution we got the same conclusion, that is the GPU is more suitable for training, and the FPGA is best in referencing, which confirms our initial hypothesis and analysis. Moreover, it proves our solution can take advantage of different high-performance devices to implement all kinds of machine learning jobs efficiently.

Also, we have a few interesting findings. For instance, when we perform the CNN case study inferencing on one image, we obtain the result shown in Table 3.10. We can see that the average

Table 3.10 Results of one image inferencing time

Experiments	CPU-E5-1620	GPU-TitanXp		FPGA-Arria10		
	Inferencing Time (us)	Inferencing Time (us)	Acceleration (GPU/CPU)	Inferencing Time (us)	Acceleration (FPGA/CPU)	Acceleration (FPGA/GPU)
1	3172	616045	0.0051	88.74	35.7	6942.4
2	5564	589114	0.0094	90.12	61.7	6536.7
3	4620	588444	0.0079	94.83	48.7	6205.3
4	3234	598652	0.0054	84.57	38.2	7079.0
5	4037	600288	0.0067	101.08	39.9	5938.8
6	4579	609913	0.0075	108.74	42.1	5609.0
Average	4201	600409	0.0070	94.70	44.4	6341.5

speed of Arria 10 is about 44.4 times faster than the CPU E5-1620 and is about 6, 342 times faster than the GPU TitanXp. In Wang et al.'s work (Wang *et al.*, 2017), FPGA was 36.1X faster than CPU. These results justify that our decision to use the FPGA to do inferencing is correct. Concerning the GPU's slight underperformance, we explain that the workload associated with the inferencing of one image is too small compared with the GPU initialization time and delay. Therefore, the total time of the GPU, which includes the initialization time and inferencing time is the longest.

Additionally, in the DNN case study, when the training sets' batch size is small, e.g., less than 32, the training speed of GPU is even slower than the training speed of CPU. Our explanation is CPU has better bandwidth and frequency than GPU, as the transfer speed of our ddr4-2400 is

19, 200 MB/s, the transfer speed of GPU which equals the PCIe Gen 3 x 16 is 15, 760 MB/s, and the max frequency of our CPU is 3, 800 MHz, the max frequency of our GPU is 1582 MHz. When the computing job is too small, although the GPU has more parallel computing cores, GPU does not have any further advantages over CPU.

3.8 Conclusion and future work

In this paper, we presented a hybrid, GPU-FPGA based design methodology for enhancing machine learning applications' performance. After carefully comparing and analyzing the characters and the structures of CPU, GPU and FPGA, the results of our investigations suggest that the GPU is more suitable for training while the FPGA is best for inferencing and a model converter is necessary when the training and inferencing frameworks are different. Therefore, to achieve higher machine learning performance, a better strategy would be to implement the training module on the GPU and the inferencing module on the FPGA.

According to the above design methodology, we implemented two machine learning systems. One is a CNN for handwriting digit recognition, and the other is a DNN regression for the estimation of the data center's PUE. The results of the two use cases confirm clearly that our hypothesis and analysis are correct. Also, it proves that our ML platform solution can take advantage of different high-performance devices to implement all kinds of machine learning jobs efficiently.

In our future work, we plan to do further investigation on the power analysis of the hybrid ML system. Besides, we will summarize the experience of model converting to identify standard rules for any model converting.

CHAPTER 4

MULTISPECTRAL IMAGE RECONSTRUCTION FROM COLOR IMAGES USING ENHANCED VARIATIONAL AUTOENCODER AND GENERATIVE ADVERSARIAL NETWORK

Xu Liu[1] , Abdelouahed Gherbi[1] , Zhenzhou Wei[2] , Wubin Li[3] , Mohamed Cheriet[1]

[1] Synchromedia Laboratory, Université du Québec (École de technologie supérieure),
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Department of Electrical and Computer Engineering, McGill University,
845 Sherbrooke Street West, Montréal, Québec, Canada H3A 0G4
[3] Ericsson Canada,
8275 Trans Canada Route, Saint-Laurent, Québec, Canada H4S 0B6

Article published on the journal « IEEE Access » in December 2020,
DOI: 10.1109/ACCESS.2020.3047074

4.1 Abstract

Since multispectral images (MSIs) have much more sufficient spectral information than RGB images (RGBs), reconstructing MS images from RGB images is a severely underconstrained problem. We have to generate colossally different information between the two scopes. Almost all previous approaches are based on static and dependent neural networks, which fail to explain how to supplement the massive lost information. This paper presents a low-cost and high-efficiency approach, "VAE-GAN", based on stochastic neural networks to directly reconstruct high-quality MSIs from RGBs. Our approach combines the advantages of the Generative Adversarial Network (GAN) and the Variational Autoencoder (VAE). The VAE undertakes the generation of the lost variational MS distributions by reparameterizing the latent space vector with sampling from Gaussian distribution. The GAN is responsible for regulating the generator to produce MSI-like images. In this way, our approach can create huge missed information and make the outputs look real, which also solves the previous problem. Moreover, we use several qualitative and quantitative methods to evaluate our approach and obtain excellent results. In particular, with much less training data than the previous approaches, we obtained comparable results on the CAVE dataset and surpassed state-of-the-art results on the ICVL dataset.

4.2 Introduction

Lights with different wavelengths have different reflection, refraction, and transmission properties. People use Multi-Spectral Images (MSIs) to record these differences. MSIs consist of several channels or bands, and each band contains the amount of radiation measured in a particular wavelength range (Ose, Corpetti & Demagistri, 2016).

From MSIs' abundant spectral information, many MSIs' applications have been developed, such as land mine detection (Makki, Younes, Francis, Bianchi & Zucchetti, 2017), satellite remote sensing (Gevaert, Suomalainen, Tang & Kooistra, 2015), medical imaging (Andersson-Engels, Johansson & Svanberg, 1994), weather forecasting (Ellrod, 1995), and interpretation of ancient documents and artworks (Baronti, Casini, Lotti & Porcinai, 1998).

Although MSIs have a wide range of applications, acquiring them is a complicated, costly, and time-consuming process, since MSIs have many bands that have to be taken one by one. Moreover, obtaining each band's data requires a specific wavelength lens to filter out other wavelength lights and to be stored in a dedicated space. Therefore, much time and storage space are consumed in changing the lens and saving each band's image. Moreover, we can synthesize RGBs with high precision from MSIs according to the Color Matching Functions (Magnusson *et al.*, 2020). From MSIs to RGBs, it is a straightforward process.

On the contrary, RGB images only have three bands: red, green, and blue. The three colors are used as the primary colors to constitute other colors (different wavelength light). Most of our daily used devices' cameras can take RGB images, which are much more convenient and cheaper to obtain. Hence, it is straightforward to consider reconstructing MSIs from RGBs directly. However, the reconstruction process is very complex and challenging.

In order to illustrate this challenge, Figure 4.1 is the schematic diagram that presents a preliminary idea of the challenge. Supposing we have a 3-bands 512×512 RGB image and a 31-bands 512×512 MS image, the range of each pixel value is from 0 to 255. According to information theory, the maximum information contained by an RGB image is $-\log 256^{512*512*3}$ (nats), while

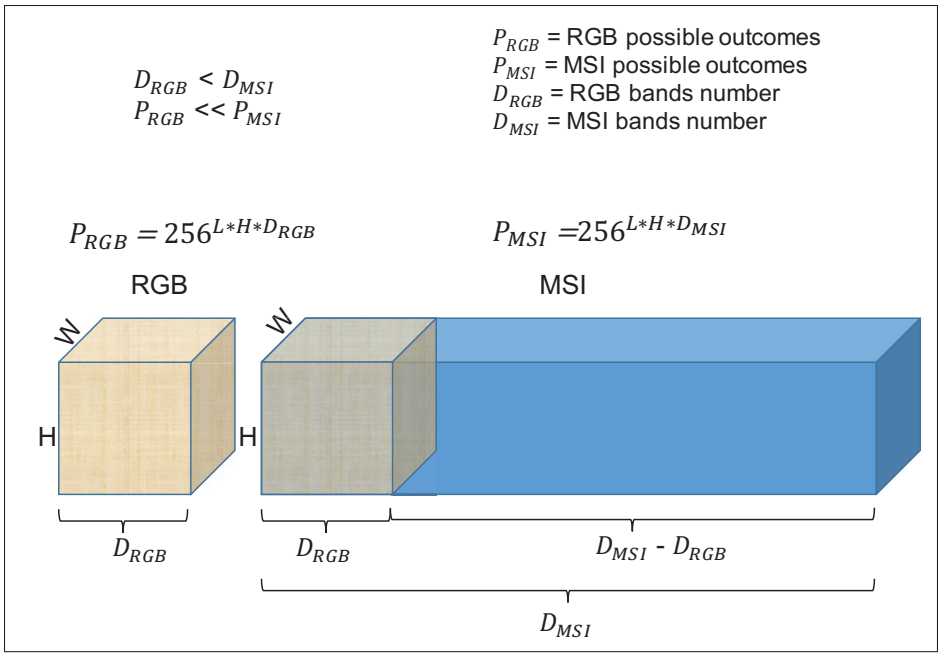

Figure 4.1 The schematic diagram of possible spectral space

the maximum information of an MS image is $-\log 256^{512*512*31}$ (nats). There is a vacancy of more than 10 times between the two spaces, which turns the reconstruction of MS images from RGB images into an extremely underconstrained problem. Compared with RGB images, MS images have a much higher spectral resolution, which may cause a difficult problem of one RGB image mapping to many MS images (Abraham, 2020).

To tackle this problem, we propose a new approach in this paper, whose fundamental concept is to replace the traditional autoencoder with the VAE when implementing the generator of the GAN and to add an L1 regulator to assist in training the generator. The VAE brings in Gaussian noise by reparameterizing the latent vector, which breaks the direct link from the input to the output. Meanwhile, with sampling from a continuous normal distribution, the generator could create infinite variational output MSI patterns. In this way, one RGB image can generate unexhausted latent vectors, which can create countless MS images. The adversary network helps the generator make real-like MSIs and the L1 regulator collapses multiple possible real-like results into one result. Following the above flow, we can acquire the MS image that we desire.

The rest of the paper is organized as follows: Section 4.3 presents some related work. Section 4.4 demonstrates the proposed approach. Next, we perform several experiments to evaluate the performance of the proposed method and compare our results with state-of-the-art results in Section 4.5. In Section 4.6, there is a brief discussion about some limitations of our approach. We summarize our current work and introduce some future work in Section 4.7. In Appendix 4.8, we provide all the detailed architectures of the neural network involved.

4.3 Related work

MS image reconstruction is not a new field. Early in 2014, Rang et al. tried to use synthesized RGBs after white balancing and the radial basis function (RBF) network to reconstruct MSIs. This method behaves well when the reflectance and illumination have a smooth spectrum. However, in the case of a spiky spectrum, the approach yields poor results. Rang et al.'s work also involves a limitation because they assumed the use of a uniform illumination to illuminate the scenes (Nguyen *et al.*, 2014).

In 2016, Arad et al. reconstructed hyperspectral images using a sparse dictionary of hyperspectral signatures and RGB projections. Although their results achieved state-of-the-art, their approach had to make a hyperspectral prior by sampling from each dataset image, restricting the fields of its application. Also, its reconstruction quality relied heavily on the scope and specificity of the hyperspectral prior (Arad & Ben-Shahar, 2016).

These approaches are based on traditional solutions. Such solutions have a common shortfall: they often have too many prerequisites, such as the equipment and the illumination. When the environment or the dataset changes, they need to retune their model's parameters. Deep learning approaches do not have this shortfall. Once the neural network design has been finished, we need to use only the new dataset to train the neural work when the dataset changes. These are data-driven approaches. Many researchers have tried to leverage this new technology to solve the MSI reconstruction problem.

Early in 2017, Zhiwei et al. proposed HSCNN based on CNN, which takes the spectrally upsampled image as input and outputs the enhanced hyperspectral images. They claimed their results significantly improved the state-of-the-art. (Xiong *et al.*, 2017)

In 2018, by removing the hand-crafted upsampling in HSCNN, Zhan et al. developed HSCNN+, which has two kinds of networks, HSCNN-R and HSCNN-D. HSCNN-R consists of several residual blocks, and HSCNN-D replaces the residual blocks by a dense block with a novel fusion scheme (Shi, Chen, Xiong, Liu & Wu, 2018). In the NTIRE 2018 Spectral Reconstruction Challenge, HSCNN-D ranked first, and HSCNN-R ranked second (Arad *et al.*, 2018).

Meanwhile, in 2018, Berk et al. proposed three models based on the Convolutional Neural Network (CNN): a generic model, a conditional model, and a specialized model. The generic model is a direct mapping from RGBs to MSIs. The others two need additional networks to estimate or classify the sensitivities. Moreover, they proved that efficiently estimating the sensitivity function and conditioning the spectral reconstruction model are useful for improving reconstruction accuracy (Kaya *et al.*, 2018). Although their approach ranked seventh in the NTIRE 2018 Spectral Reconstruction Challenge, they claimed their solution to be the most efficient, with the lowest number of layers and shortest runtime.

Meanwhile, in 2018, Xiaolin et al. suggested utilizing K-means classification to separate an RGB image into different classes according to their spectrums and then applying backpropagation neural networks (BPNNs) to reconstruct the corresponding hyperspectral image. Their approach had to establish a mapping between the RGB and the MSI for each class as a foundation (Han, Yu, Xue & Sun, 2018).

Furthermore, at the 2019 CVPR workshop, Kin et al. demonstrated a way of directly reconstructing MSIs from RGBs by using conditional GAN. Since their method is purely data-driven, it could easily lead to hallucinatory results (Gwn Lore, Reddy, Giering & Bernal, 2019).

Meanwhile, some contributed solutions appeared at the NTIRE 2020 Challenge on Spectral Reconstruction from an RGB Image (Arad *et al.*, 2020). For example, Jiaojiao et al. proposed a

novel adaptive weighted attention network for spectral reconstruction. They stacked multiple dual residual attention blocks to build the backbone of the approach. Their entries obtained first rank on the "Clean" track and third place on the "Real World" track. (Li, Wu, Song, Li & Liu, 2020)

Although the above mentioned papers have achieved some progress in reconstructing MSIs from RGBs, almost all the methods except Kin et al.'s are based on static and dependent neural networks. Since there are no random elements in their neural networks, their models cannot generate new information or previously unseen distributions. Thus, they failed to answer the crucial question: how to supplement the lost information between RGBs and MSIs with their approaches. However, in the following sections, we will answer it with our work.

4.4 The proposed method

4.4.1 The problem analysis and the proposed solution

From the previous introduction, we know that the tremendous challenge of reconstructing MSIs from RGBs is how to use a small spectral space to represent an ample spectral space. It is a severely underconstrained problem that will result in the metamerism phenomenon (Palmer, 1999). The main idea of metamerism is that different multispectral distributions map to the same RGB distribution. Thus, the metamerism prevents the synthesis of the correct MSIs from the RGBs, since different MSI labels with the same RGB input may cause the gradient to descend in different directions and lead to problematic convergence. Figure 4.2 (a) is the standard solution based on Auto Encoder, one RGB pixel maps to one latent vector; and the latent vector maps to many MS pixels, which make the model training hard to converge. It is also very similar to a multi-directional tug of war; as different teams of people exert force in different directions, the center of the rope swings in different directions.

Since the problem has been described clearly, the solution is also apparent: try to create more variational inputs to map to multiple outputs to reduce the input entanglement effect.

There are two places to add variations. One is on the input side, and the other is in the latent space. However, if we add the variations on the input side, they are easily ignored. The explanation is that the structure of the Auto Encoder is good at denoising, since it compresses the input information, including the variations, and the minor variations are easily thrown out in the compression process. Several authors also have reported this phenomenon (Isola, Zhu, Zhou & Efros, 2016; Mathieu, Couprie & LeCun, 2015). Meanwhile, we have verified this side effect in our experiments. However, if we insert the noises (variations) in the latent space, they escape the lossy compression process and are enlarged directly by the decoder. To this end, we strive to add variations to the latent vector.

Figure 4.2 (b) depicts our proposed solution. We use the Variational Auto Encoder instead of the typical Auto Encoder. In this way, one RGB pixel still maps to one latent vector in the first phase. However, this latent vector will be re-parametered into several different latent vectors by later random sampling from the normal distribution. The number of re-parametered latent vectors can be unlimited, greater than the limited number of possible MSI pixels. Therefore, each input with one random latent vector can map to at least one output. The re-parameterization step turns the input space from limited into unlimited. Furthermore, with the re-parameterization step, the proposed approach is equivalent to building a bridge between the two deterministic networks, encoder and decoder, and making the gradients backpropagate from output to the input. In this way, the previous underconstrained problem can be solved.

4.4.2 The detailed implementation of the proposed method

4.4.2.1 The model's architecture

According to the previous analysis, we designed a special GAN, as shown in Figure 4.3. The most significant difference from the typical GAN is that we used the VAE to substitute the AE. The AE has a fixed latent space, which means that the model's parameters are fixed when the training finishes. One input can lead to only one specific output. Because of the fixed neural network, when the metamerism phenomenon occurs, different labels with the same input cause

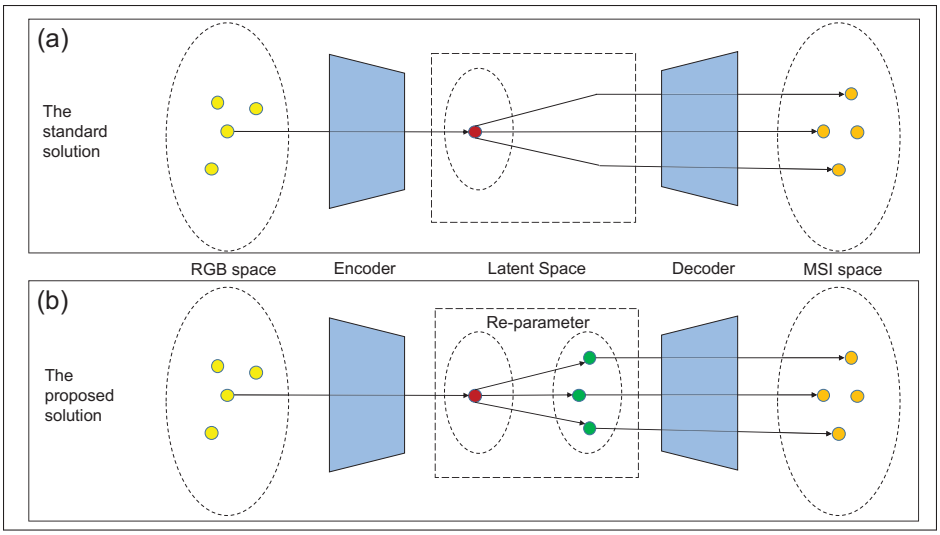

Figure 4.2 The comparison of the standard solution and the proposed solution

the gradient to descend in different ways and make the training process challenging to converge. It is difficult to learn the right mapping between the input and the output.

Figure 4.3 shows that the proposed approach consists of three main parts: encoder, re-parameter, and decoder. The encoder is a neural network that compresses an input sample RGB_1 into a hidden representation μ-mean vector and a σ-standard deviation vector. And ϕ stands for the weights and biases. We denote the encoder by $q_\phi(z|rgb)$. And then, we re-parameterize the latent space vector z with Equation (4.1). In Equation (4.1), ϵ is sampled from the normal distribution $\mathcal{N}(0, I)$ and supplies the variations to generate diverse latent space vectors "$z_1, z_2, ..., z_n$". The decoder is another neural network that generates possible MSI distributions "$MSI_1{}', MSI_2{}', ..., MSI_n{}'$" with varied $\epsilon_n s$ "$\epsilon_1, \epsilon_2, ..., \epsilon_n$", and θ denotes the weights and biases. We denote the decoder with $p_\theta(msi|z)$.

$$
\begin{aligned}
z &= \mu + \sigma \odot \epsilon \\
\epsilon &\sim \mathcal{N}(0, I)
\end{aligned}
\tag{4.1}
$$

From Equation (4.1), it can be seen that all the variations of latent space vector z are from sampling the normal distribution. The normal distribution can make the VAE latent space have a continuous variation compared with the AE fixed latent space. An input produces different latent vectors, and the decoder creates corresponding variational outputs with these variations. In this way, each input is tagged with a random Gaussian noise number label and re-parameterized into a unique latent vector; and one latent vector generates one particular output. Besides, the Gaussian distribution space is infinite, which guarantees that one latent vector can map to at least one output. The latent space will finally be disentangled.

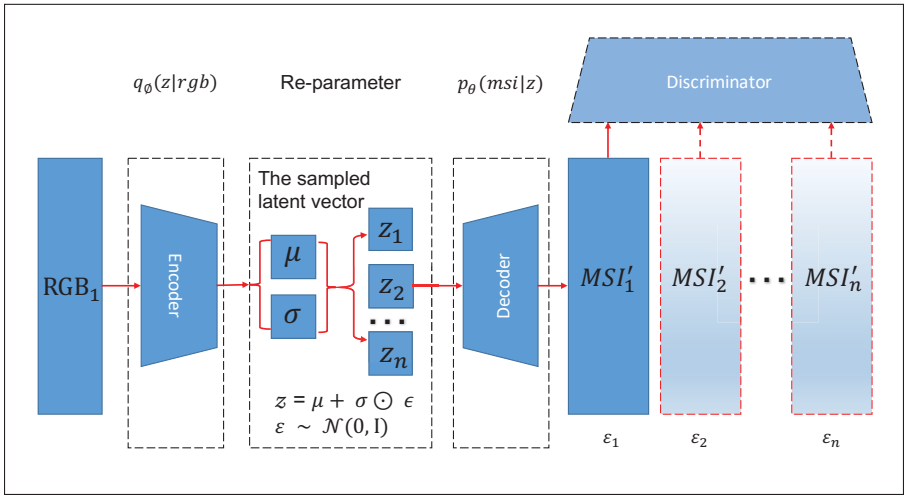

Figure 4.3 The detailed implementation of the proposed method

4.4.2.2 The model's Training

As noted above, our approach has three main parts: encoder, re-parameter, and decoder, which constitute the generator. To train the generator, we use the adversary network and Kullback–Leibler divergence (KL divergence) together.

Specifically, in order to get the generator, we designed 3 losses to train the model: Loss of VAE-$\mathcal{L}_{VAE}$, Loss of discriminator-$\mathcal{L}_D$, and Loss of GAN-$\mathcal{L}_{GAN}$.

$$\mathcal{L}_{VAE} = -\mathbb{E}_{z\sim q_\phi(z|rgb)}[\log p_\theta(msi|z)] + \beta\mathbb{KL}(q_\phi(z|rgb)||p_\theta(z|msi)) \tag{4.2}$$

$$\mathbb{E}_{z\sim q_\phi(z|rgb)}[\log p_\theta(msi|z)] = \|MSI - MSI'\|_1 \tag{4.3}$$

Equation (4.2) is the definition of $\mathcal{L}_{VAE}$, which contains a negative log-likelihood reconstruction loss, a KL divergence regularizer and a hyperparameter β. Equation (4.3) is the implementation of the reconstruction loss, and here we use the Mean Absolute Error (MAE) to calculate the error between the generated MSI and the original MSI. If the decoder fails to re-build the MSIs well, this loss becomes large. In this way, this loss helps the decoder learn to reconstruct the MSIs.

Meanwhile, the KL divergence measures how much information is lost when using $q_\phi(z|rgb)$ to represent $p_\theta(z|msi)$. It also shows how close $q_\phi(z|rgb)$ is to $p_\theta(z|msi)$. In the VAE, $p_\theta(z|msi)$ respects the standard normal distribution $N(0, I)$. The encoder will receive a penalty in the loss if the output representations $p_\theta(z|msi)$ are different from those from the standard normal distribution. Meanwhile, this regularizer tries to keep the representations $p_\theta(z|msi)$ of each datapoint sufficiently different. Without the regularizer, the encoder could learn to cheat and give each input RGB pixel the same representation in a different Euclidean space region (Altosaar, 2020).

Furthermore, β is the regularization coefficient that tunes the available ratio of negative log-likelihood reconstruction loss and KL divergence. Higher β means the KL divergence will take more role, which will bring in more variations but more noise (Higgins *et al.*, 2016).

$$\mathcal{L}_{GAN} = \mathbb{E}_{rgb\sim p_{data}(rgb)}[\log(1 - D(G(rgb)))] \tag{4.4}$$

In addition to Equation (4.2), we use an adversarial network to train the generator. Equation (4.4) defines the loss of generator. It works like other GANs by trying to cheat the discriminator and letting the discriminator think the generated MS images are real.

$$\mathcal{L}_D = \mathbb{E}_{msi \sim p_{data}(msi)}[\log D(msi)] + \mathbb{E}_{rgb \sim p_{data}(rgb)}[\log(1 - D(G(rgb)))] \tag{4.5}$$

Equation (4.5) represents the loss of the discriminator, which consists of two log-likelihood parts. The former tries to use the real MSI to train the model, and the latter allows the model learn fake examples from the results of the generator.

Combining all the above losses, we obtain the total loss $\mathcal{L}_{Total}$ (4.6). There are 2 hyper-parameters, β, γ, which have different impacts on the training process. β tunes the ratio of KL divergence, γ adjusts the GAN's functional percentage. If we want the KL divergence to have more effects, we need to increase the value of β; and if we want to use the GAN more to train the generator, we need to tune up the γ. GAN and KL divergence have different advantages and disadvantages for training the model. We will give a detailed discussion and analysis in the Section 4.6.

$$\mathcal{L}_{Total} = \|MSI - MSI'\|_1 + \beta\mathbb{KL} + \gamma\mathcal{L}_{GAN} \tag{4.6}$$

4.5 Experiments

To thoroughly evaluate our approach, we selected two classical datasets: CAVE (Yasuma, Mitsunaga, Iso & Nayar, 2008) and ICVL (Arad & Ben-Shahar, 2016). The detailed experiment description and results are in the following sections.

4.5.1 Experiment environment

We list the hardware devices used in Table 4.1 and the software involved in Table 4.2.

Table 4.1 The hardware list
Taken from Liu et al. (2018, p. 105)

Device	Type	Number
CPU	Intel(R) Xeon(R) CPU E5-1620 v4 @ 3.50GHz	1
Memory	Samsung DDR4 8g	4
Solid State Disk Drive	INTEL SSDSC2BB48 480g	1
Mechanical Hard Disk Drive	WDC WD40EFRX-68N 4TB	1
GPU	NVIDIA TITAN Xp	1

Table 4.2 The software list

Software	Version
Ubuntu	16.04.6 LTS (Xenial Xerus)
Python	3.6.7
Tensorflow-GPU	2.1.0
CUDA	10.2.89

In this project, our graphic card is NVIDIA TITAN Xp. Its architecture is Pascal and has 3840 parallel computing cores. Each core has a 1582 MHz frequency. Its memory is made of 12 GB GDDR5X, and its speed achieves 11.4 Gbps. The bandwidth attains a high point of 547.7 GB/s. According to our previous research results, the GPU's training speed is about 10 times faster than the CPU's (Liu *et al.*, 2018a, 2019b).

4.5.2 The CAVE dataset

4.5.2.1 The dataset introduction and the hyperparameter setting

Columbia University Computer Vision Laboratory made the CAVE dataset. The CAVE dataset has 32 indoor scenes, including 5 categories: stuff, skin and hair, paints, food and drinks, real and fake. Moreover, each image consists of a 31-band MS image and a 3-band RGB image. The 31 bands cover visible light from 400nm to 700nm at 10nm steps. The spatial resolution of each band is 512×512 pixels. (Yasuma *et al.*, 2008)

First, we convert the CAVE dataset from 32 (scenes) × 34 (bands) png images into a 32 (samples) × 512 (height) × 512 (width) × 34 (band-columns) numpy array, where the 1-31 columns map to the 400nm-700nm bands. Meanwhile, the 32-34 columns match the R, G, B bands individually.

Then, we split the dataset into two equal-size groups according to the image's index in the dataset. Images with odd indexes, 1, 3, 5, ..., 31, are put in one group; images with even indexes, 2, 4, 6, ..., 32, are put in another group. We rotate these two groups to be the training dataset and the evaluating dataset to thoroughly verify our approach. Generally speaking, more training data often lead to a higher performance model. Andrew Ng has given a detailed explanation with Figure 4.4 in his book "Machine learning yearning" (Ng, 2017). So, the most frequent ratio of training and evaluation is 80%:20% or 70%:30%. However, the more training data required, the fewer application domains the approach has. We choose a more challenging dataset split ratio, 50%:50%. Only a few works, like Kin et al.'s (Gwn Lore *et al.*, 2019), have chosen this kind of split ratio to the best of our knowledge. Moreover, the approach of Kin et al. also contains a GAN loss like ours. Therefore, we use their work as the primary baseline. We also compare our results with Arad et al.'s and Berk et al.'s, since Arad et al. claimed their results are state-of-the-art (Arad & Ben-Shahar, 2016) and Berk et al. claimed theirs is the first successful estimation of the spectral data from a single RGB image captured in unconstrained settings (Kaya *et al.*, 2018).

To thoroughly evaluate our method and compare it with the previous baseline, we also performed bi-directional translations between RGBs and MSIs and conducted qualitative and quantitative evaluations, respectively.

Furthermore, we normalized the data from the range [0.0-1.0] to [−1.0-1.0] before training and recovered the image data to [0.0-1.0] after training. After many trials, we found that the best setting to train the generator: $\beta = 0$, $\gamma = 10$, the training epoch is 300, and the optimizer is Adam with the learning rate $1e^{-4}$.

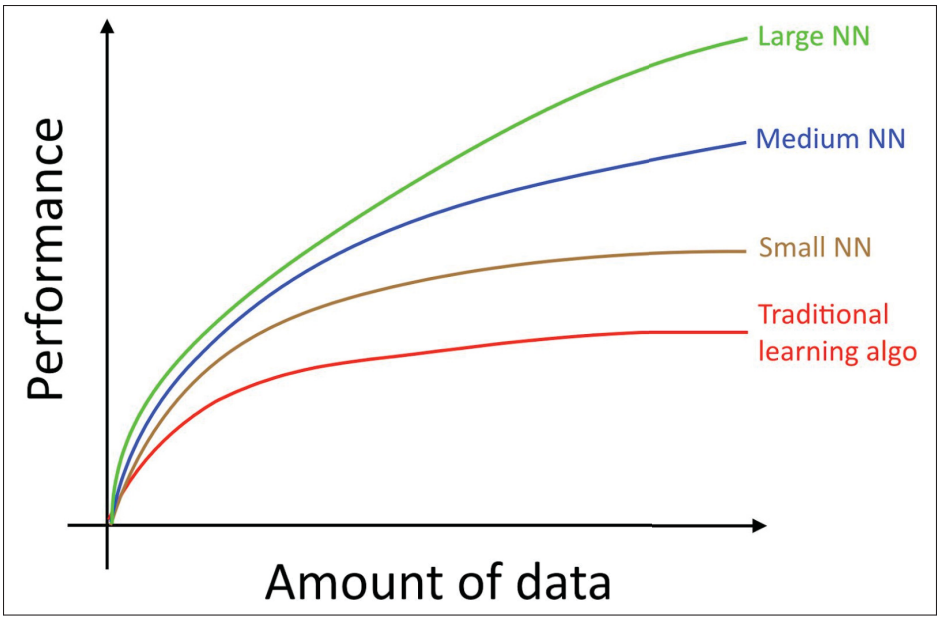

Figure 4.4 The relation between model performance and training data
Taken from Ng et al. (2021, Website)

4.5.2.2 The qualitative evaluation

In this subsection, the qualitative results of the two phases: RGB to MSI and MSI to RGB, are shown. Since we chose Kin et al.'s work (Gwn Lore *et al.*, 2019) as the baseline, all the selected images and training configurations are the same as theirs.

Figure 4.5 shows the result of the RGB to the MSI. We chose image "beads" and selected 5 spectral bands to reconstruct the MSI from the RGB image. To make the paper's layout look tidy, we put the ground truth input RGB image in Figure 4.6 top left. Moreover, we selected the same five bands as Kin et al. did to compare the results equally. The five bands are between 400nm and 700nm with approximately equal intervals. Since it is hard to identify nearby wavelength lights' subtle differences with the naked eyes, the five bands are enough to demonstrate the qualitative evaluation of spectral reconstruction. Furthermore, it is a prevalent way to select several bands to show qualitative results (Arad & Ben-Shahar, 2016; Kaya *et al.*, 2018). Otherwise, in the

quantitative evaluation section, we demonstrate the full RMSE varieties against the whole spectrum (400nm-700nm) reconstruction results in Figure 4.7.

Besides, we got Error maps using the prediction image minus the ground truth image and then pseudocolored the error images with the "jet" colormap. Red, green, and blue indicate negative, zero, and positive errors, respectively. After comparing our results with Kin et al.'s work, we found that our model behaves better than theirs, especially in the 410nm, 550nm, and 590nm bands.

Moreover, we can quickly reconstruct the RGB images from MS images with a similar neural network when we reverse the input and output of the VAE generator and make some small changes, such as resetting the input and output size. Figure 4.6 is the result of reconstructing the RGB from the MSI. We find that the reconstructed RGB images and ground truth images are too close to judge with the naked eyes. Furthermore, the reconstruction behaves better than Kin et al.'s work (Gwn Lore *et al.*, 2019).

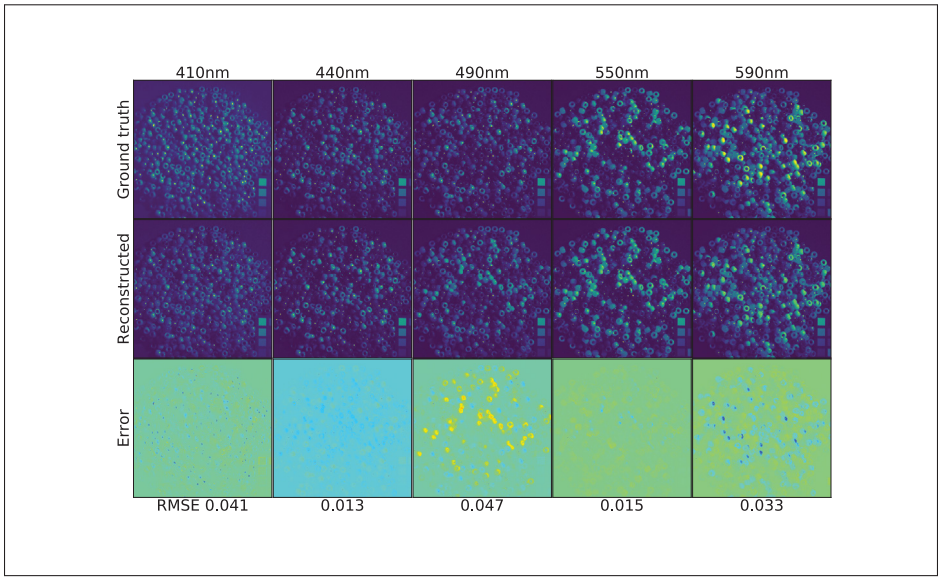

Figure 4.5 The reconstruction of five selected spectral bands using an RGB image (The input RGB is the top left image of Figure 4.6.)

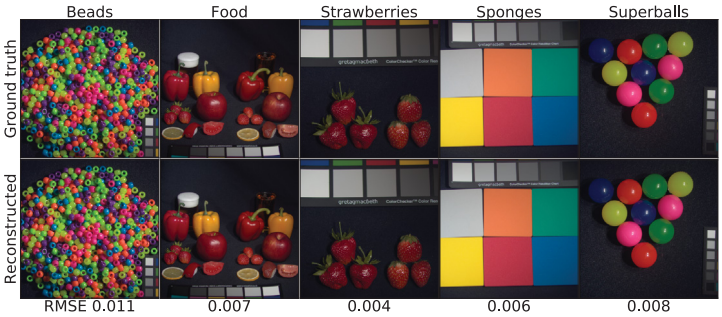

Figure 4.6 The reconstruction of five selected RGB images using MS images

4.5.2.3 The quantitative measurement

To compare our results fairly with the related work, we chose three classical quantitative metrics: Root Mean Square Error (RMSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM), to evaluate our model's performance.

As stated above, we practiced with 16 MS and RGB images with the odd number position to train the model and another 16 scenes with the even number position to test the model, and vice versa. We performed the previous process 3 times and then calculated the average value. Furthermore, the reconstruction experiment is bi-directional, which means that we measure the results of converting both the RGB to the MSI and the MSI to the RGB. In order to highlight the comparison easily, we include our results with Berk et al.'s (Kaya *et al.*, 2018), Kin et al.'s(Gwn Lore *et al.*, 2019), and Arad et al.'s (Arad & Ben-Shahar, 2016) in the Table 4.3 and Table 4.4, respectively.

From Table 3, it can be seen that Arad et al. have the lowest RMSE. However, our results are very close to theirs and rank second. For this result, we provide the following explanation.

First, the ratio of splitting the CAVE dataset for training and evaluation is not explicit in Arad et al.'s paper. As noted above, in most cases, more training data usually yield a higher performance model.

Second, before reconstruction, their method had to make a hyperspectral prior by sampling from each image. The sampling ratio of the CAVE dataset is 3.8% of each image. Their approach needs to gather information from the whole dataset, including the training dataset and the test dataset, which will help their model improve its performance. However, the requirements for more information will limit the application scope of their approach.

On the contrary, the ratios of Kin et al. and our approaches are 50% for training and 50% for testing. The training dataset and the testing dataset are entirely isolated, which means that the model knows nothing about the test dataset when doing the testing. So, our approaches' application fields will be more prevalent. Given the above analysis, it is not fair to compare our and Kin et al.'s results with Arad et al.'s results. By comparing our results with Kin et al.'s, it can be seen that the RMSE of reconstructing the MSI from the RGB has been reduced by 29%, and the RMSE of reconstructing the RGB from the MSI has been reduced by 66%.

Table 4.3 The average RMSE, PSNR and SSIM of reconstructing MSI from RGB

Metrics	**Berk**	**Kin**	**Arad**	**Ours**
Approach	CNNs	cGANs	Sparse coding	**VAE-GAN**
Ratio of training and testing	N/A	50%:50%	N/A	**50%:50%**
RMSE~(0-255)	N/A	8.0622	5.4	**5.741**
RMSE~(0-1)	0.038	N/A	N/A	**0.023**
PSNR	28.78	N/A	N/A	**34.00**
SSIM	0.94	N/A	N/A	**0.98**

Table 4.4 The average RMSE, PSNR and SSIM of reconstructing RGB from MSI

Metrics	**Berk**	**Kin**	**Ours**
Approach	CNNs	cGANs	**VAE-GAN**
Ratio of training and testing	N/A	50%:50%	**50%:50%**
RMSE~(0-255)	2.55	5.649	**1.943**
RMSE~(0-1)	0.038	N/A	**0.0076**
PSNR	28.78	N/A	**42.96**
SSIM	0.94	N/A	**0.99**

Besides, we selected the same six images as Kin et al. did in their paper (Gwn Lore *et al.*, 2019): "Beads," "Food," "Strawberries," "Sponges," "Superballs," and "chart & Stuffed Toy" to calculate their RMSEs of MSI 31 bands reconstruction and draw the curves of RMSEs against the wavelengths in the Figure 4.7.

Figure 4.7 reveals that the model behaves with different prediction abilities when facing different scenes. Moreover, the model performs better when reconstructing a band image whose wavelengths are in the middle range. Our explanations are as follows. From Figure 4.8 CIE 1931 standard observer color matching functions (Amara, M. et al., 2018), we notice that most of R, G, and B components are distributed principally from 450nm to 650nm wavelengths and a few ingredients are allocated to near the two ends of the spectrum. A color pixel constituted of R, G, and B bands holds little information about the two ends of the spectrum, which may cause the low reconstructing ability at the two ends of the spectral zone.

4.5.3 The ICVL dataset

4.5.3.1 The dataset introduction and the hyperparameters setting

The Ben-Gurion University interdisciplinary Computational Vision Lab made the ICVL dataset. The latest version contains 201 images (Most scenes are outdoor.) taken by a Specim PS Kappa DX4 hyperspectral camera. Moreover, each image has 1392×1300 spatial resolution over 519 spectral bands ($400 - 1,000nm$ at roughly $1.25nm$ increments) (Arad & Ben-Shahar, 2016).

In the experiment, we used the reduced ICVL dataset supplied by Arad et al. They reduced the 519 bands to 31 bands of roughly 10 nm in 400–700 nm for two reasons: reducing computational cost and facilitating comparison to previous benchmarks that employ this kind of representation (Arad & Ben-Shahar, 2016). Moreover, we calculated the RMSE between the RGB image generated by 519 bands and the RGB image generated by 31 bands. The RMSE is about 0.00014, which means the two formats have few differences.

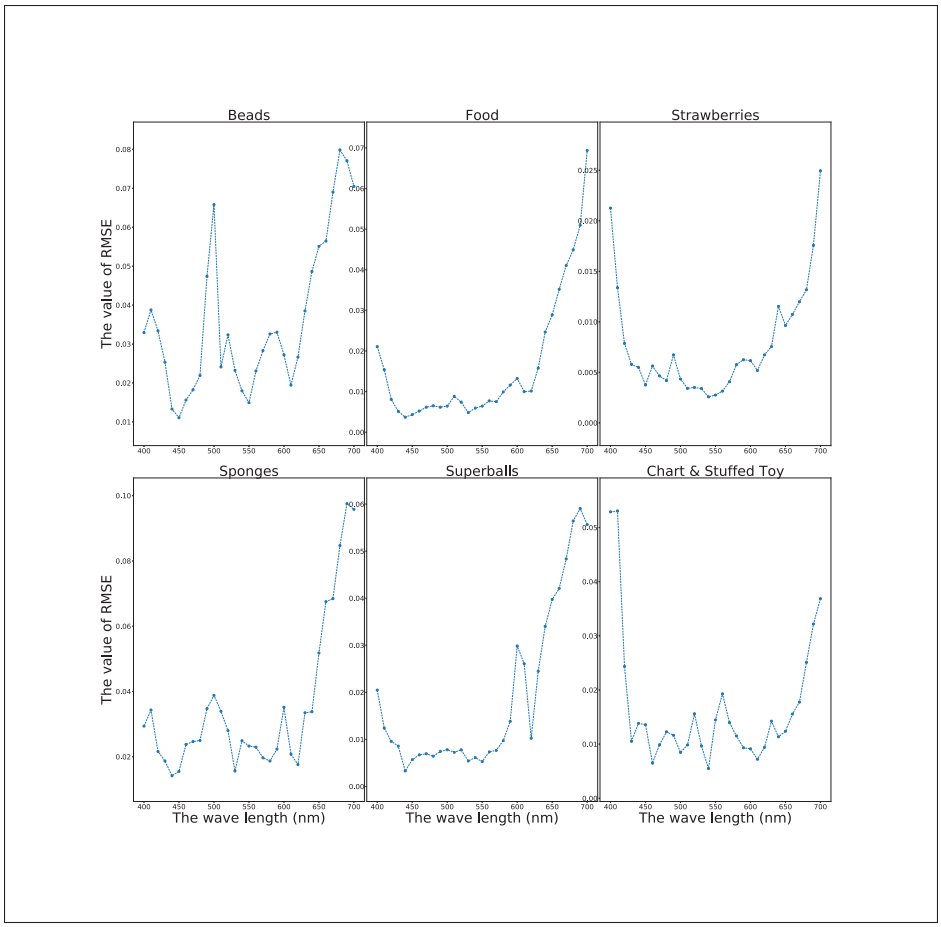

Figure 4.7 The RMSE curves of MSI reconstruction

To evaluate our approach thoroughly and non-overlappingly, we first randomized the order of 201 images and then divided them into 6 groups. Each of the first 5 groups, 0, 1, 2, 3, 4, has 32 images. However, the sixth group, group 5, has 41 images. We rotated one of the first five groups, $0-4$, as the training dataset and the remain five groups as the testing dataset. For example, if we take group 1 as the training dataset, groups 0, 2, 3, 4, and 5 are the testing dataset, which means that the training and testing dataset splitting ratio reaches $32:169$, 32 images for training and 169 images for testing.

We split each training dataset of 32 images into two equal subgroups according to the images' index in the dataset. Images with odd indexes, 1, 3, 5, ..., 31, are put in one subgroup, whereas

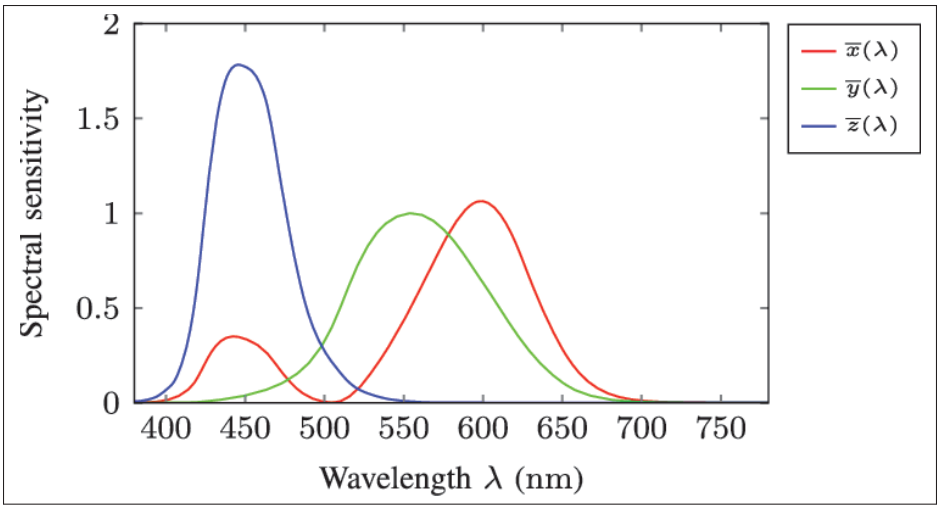

Figure 4.8 CIE 1931 color matching functions
Taken from Amara et al. (2018, p. 9)

images with even indexes, 2, 4, 6, ..., 32, are put in another subgroup. We rotate these two subgroups to be the training dataset and the validating dataset.

Furthermore, we reduce the 32 training images' spatial resolution from 1392×1300 to 512×512 by sampling randomly one of the four parts (top left, top right, bottom left, bottom right) of the original image. The final training dataset consists of 32 images with a spatial resolution of 512×512. Meanwhile, the testing dataset includes 169 images with a spatial resolution of 1392×1300. To the best of our knowledge, we are the first to use so few images to train the ICVL model.

To thoroughly evaluate our method, we also performed bi-directional translations between RGBs and MSIs and conducted quantitative and qualitative evaluations. Besides, we selected three related excellent works, Zhiwei et al. (Xiong *et al.*, 2017), Arad et al. (Arad & Ben-Shahar, 2016), and Berk et al. (Kaya *et al.*, 2018), for comparison. Among them, Zhiwei et al.'s HSCNN claimed their results were state of the art.

Furthermore, we normalized the data from the range [0.0-1.0] to $[-1.0\text{-}1.0]$ before training and recovered the image data to [0.0-1.0] after training. After many trials, we found the best setting

to train the generator: $\beta = 0$, $\gamma = 10$, the training epoch is 300, and the optimizer is Adam with the learning rate $1e^{-4}$.

4.5.3.2 The qualitative evaluation

In this subsubsection, we continue to demonstrate the qualitative results of the two reconstruction phases: RGB to MSI and MSI to RGB. We emphasized again that the training data is completely isolated from the testing data.

Since Arad et al. created the ICVL dataset and claimed their results are state-of-the-art, we chose their results as the baseline. Figure 4.9 is made like Arad et al.'s Figure 4. We selected the same two scenes and the same three bands (460nm, 540nm, and 620nm) to demonstrate our model's reconstruction performance. To make the paper's layout look tidy, we put the two ground truth input images in the top left first image "prk_0328-1025" and top left second image "BGU_0403-1419-1" of Figure 4.10.

Moreover, we got Error maps using the prediction image minus the ground truth image and then pseudocolored the error images with the "jet" colormap. Red, green, and blue indicate negative, zero, and positive errors, respectively.

When comparing Figure 4.9 and Figure 4.10 with Figure 4.5 and Figure 4.6, a phenomenon can be easily observed; the same approach works better on the ICVL dataset than on the CAVE dataset. Our explanation is that most of the CAVE dataset images contain large dark background areas, which provide little information for training the model.

4.5.3.3 The quantitative measurement

To compare our work fairly with the previous work, we chose four classical quantitative metrics: Root Mean Square Error (RMSE), Normal or Relative Root Mean Square Error (nRMSE or rRMSE), Peak Signal-to-Noise Ratio (PSNR), and Structural Similarity Index (SSIM), to evaluate our model's performance.

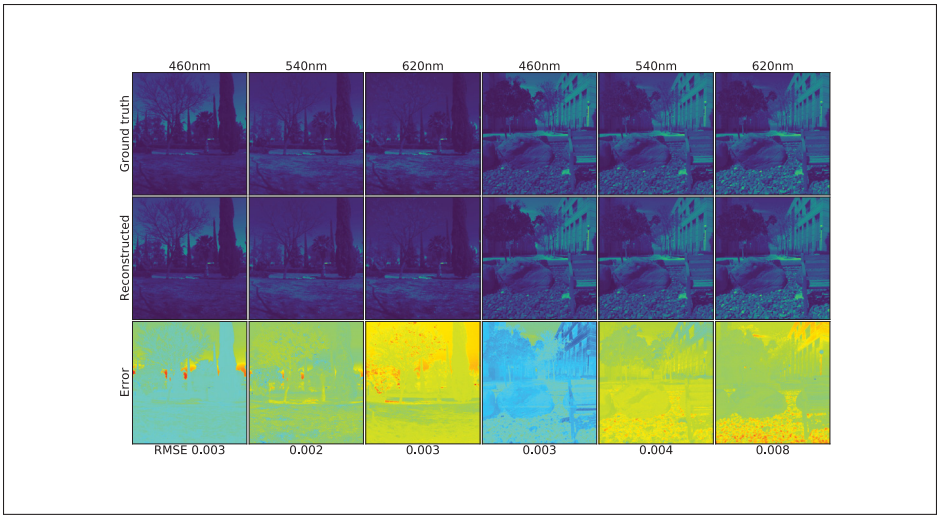

Figure 4.9 The reconstructions of three selected spectral bands images of two scenes (The two input RGBs are the top left two images of Figure 4.10.)

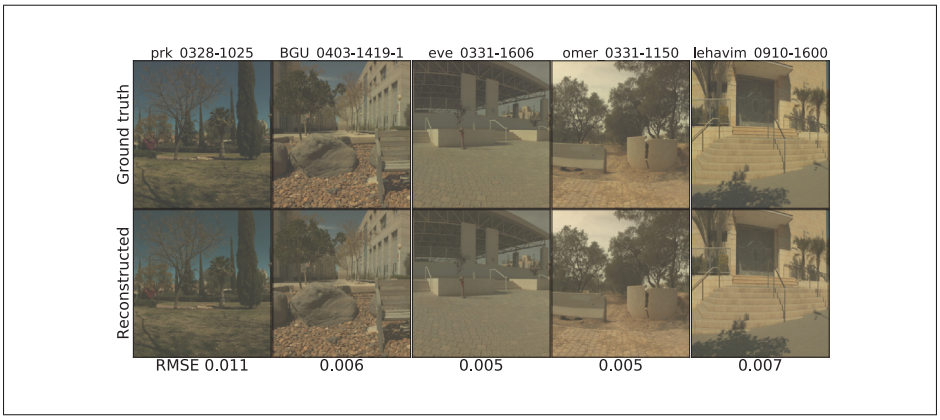

Figure 4.10 The reconstruction of five selected RGB images using MS images

Moreover, we prepare two experiments and several comparisons with the state-of-the-art to demonstrate our approach's superiority.

In the first experiment, we compared our approach, VAE-GAN, with the two state-of-the-art studies: Arad et al.'s "sparse coding" and Zhiwei et al.'s HSCNN. For equal comparison, we did some processes on the ICVL dataset.

According to the report from Arad et al., the whole experiment was conducted on the "complete set" of 102 images. After double-checking, we found that image "lst_0408-0924" does not exist in their supplied dataset (Arad & Ben-Shahar, 2020). So, we deleted image "lst_0408-0924" from the complete set. The number of the complete set images thus became 101. Moreover, 59 domain-specific images belong to the 101 images. There are 5 domains, or subsets: Park, Indoor, Urban, Rural, and Plant-life. Arad et al. selected one image from a set for testing and the rest of the set images to train the dictionary. Furthermore, they repeated the previous step until every image in the set had been chosen for testing. In the last step, they calculated the average relative RMSE (rRMSE). In this way, Arad et al. achieved better results in the domain-specific subsets than with the complete set (Arad & Ben-Shahar, 2016).

Zhiwei et al.'s HSCNN performed in a more generalizable way. They used a total of 200 images, including the complete set to train a CNN model with 141 images, excluding the domain-specific subsets. They tested the model on the 59 domain-specific images. Then, they trained another model with 159 images, excluding the non-domain-specific images in the complete set, and tested the obtained model on these 41 images. In this way, the images for training and testing were rigorously separated, and HSCNN eliminated the domain-specific restriction imposed in sparse coding (Xiong *et al.*, 2017).

Our dataset splitting is similar to Zhiwe et al.'s but has more challenges. We used the up to date dataset, which contains 201 images. Moreover, we used only the 100 images, excluding the 101 complete set's images to train the model. We then tested the model on the 101 complete set images, including the 59 domain-specific images. In this way, we isolated the training and testing images rigorously, eliminated the domain-specific restriction, and reduced the number of training images.

Table 4.5 compares the results of the above three approaches. We find that our approach, VAE-GAN, surpasses the two other approaches in the complete set and most of the subsets in the table. Moreover, this record was created with the lowest ratio of training and testing.

Table 4.5 The comparison of RGB to hyperspectral conversion

	Arad et al. (Sparse coding) Train:Test N/A	**Zhiwe et al. (HSCNN) Train:Test 141:59**	**Ours (VAE-GAN) Train:Test 100:101**
Data Set	**rRMSE**	**rRMSE**	**rRMSE**
Complete set	0.0756	0.0388	**0.0348**
Park subset	0.0589	0.0371	**0.0334**
Indoor subset	0.0507	0.0638	**0.0463**
Urban subset	0.0617	0.0388	**0.0322**
Rural subset	0.0354	**0.0331**	0.0381
Plant-life subset	0.0469	0.0445	**0.0426**
Subset average	0.05072	0.04346	**0.03852**

In the second experiment, we handled an extreme challenge to thoroughly explore our approach's full capacity. We did not use the above tactic of splitting the dataset. Rather, we used the splitting tactic introduced in subsubsection 4.5.3.1.

We randomized the order of 201 images and then divided them into 6 groups. Each of the first 5 groups, 0, 1, 2, 3, 4, has 32 images. The sixth group, group 5, has 41 images. We rotated one of the first five groups, $0-4$, as the training dataset and the remain five groups as the testing dataset. For example, if we take group 1 as the training dataset, groups 0, 2, 3, 4, and 5 are the testing dataset, which means the training and testing dataset splitting ratio reaches $32:169$, 32 images for training and 169 images for testing.

Moreover, we split each training dataset 32 images into two equal subgroups according to the image's index in the dataset. Images with odd indexes, 1, 3, 5, ..., 31, were put into one subgroup, and images with even indexes, 2, 4, 6, ..., 32, were put in another subgroup. We rotated these two subgroups to be the training dataset and the validating dataset.

Furthermore, we reduced the 32 training images' spatial resolution from 1392×1300 to 512×512 by sampling randomly one of the four parts (top left, top right, bottom left, bottom right) of the original image. The final training dataset consists of 32 images with a spatial resolution of

512×512. And the testing dataset includes 169 images with a spatial resolution of 1392×1300. To the best of our knowledge, we are the first to use so few images to train the ICVL model.

With the above training and testing tactic, we got the experimental results of RGB to MSI conversion and MSI to RGB conversion, respectively; these results are listed in Table 4.6. This is the first novel experiment with this kind of dataset splitting to achieve the best results to our knowledge. In comparison with previous CAVE results, Table 4.3 and Table 4.4, it can be quickly seen that the ICVL results are much better than the CAVE results, which is consistent with the previous ICVL Qualitative results' analysis.

Table 4.6 The bi-directional conversion results of VAE-GAN on ICVL dataset

VAE-GAN Train:Test ——32:169					
Metrics	**PSNR**	**SSIM**	**RMSE**	**rRMSE**	**RMSE_INT**
RGB to MSI	43.388	0.999	0.008	0.037	1.921
MSI to RGB	46.809	1.000	0.005	0.013	1.349

Besides, we selected three images: "prk_0328-1025", "BGU_0403-1419-1", and "eve_0331-1606" to calculate their MSI reconstruction RMSE of 31 bands individually and drew their RMSE curves in the Figure 4.11. "prk_0328-1025" and "BGU_0403-1419-1" appeared in Arad et al.'s paper (Arad & Ben-Shahar, 2016), and Zhiwei et al. adopted "eve_0331-1606" as an example (Xiong *et al.*, 2017).

It is worth noting in considering Figure 4.11, that the model behaves worse in the long-wavelength range. Our explanation of that phenomenon is that because the three images were taken outdoors in daylight, the light with long wavelengths was rapidly immersed in the background noise. With a regular camera, it is hard to capture the subtleties of long-wavelength light. Thus, because the RGB image contains few features of long-wavelength light, which is the chief reason for the poor MSI reconstruction performance in the long-wavelength spectral zone.

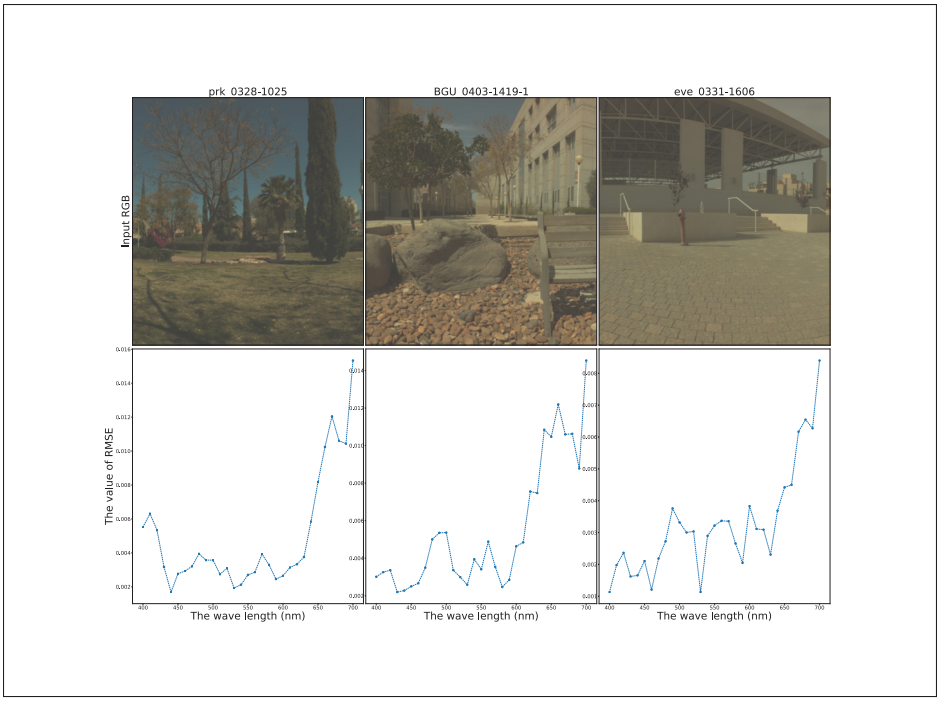

Figure 4.11 The RMSE curves of MSI reconstruction

4.6 Limitations

VAE-GAN has many hyperparameters. Therefore, there are numerous means of optimizations.

For example, we used two losses, KL-divergence and GAN, to train the model; however, we did not know which is better. β is the key hyper-parameter, which can tune the functional percentage of the two losses. We performed two interesting experiments to investigate how the model's reconstruction ability alters with β's change on the CAVE and ICVL datasets, respectively.

Table 4.7 is the experiment data for the RMSE change against the β variations on the CAVE dataset, and Table 4.8 is the data for the RMSE change against the β variations on the ICVL dataset. Furthermore, for convenient visual and comparison purposes, we included the above two tables' data in Figure 4.12.

From them, we can find a common rule: The model's prediction performance will decrease as the value of β increases, which indicates the GAN loss has a better efficiency on training the generator to synthesize real-like MSIs or RGBs.

Table 4.7 The CAVE-RMSE changes against the β variations

β	**PSNR**	**SSIM**	**RMSE (0-1)**	**RMSE (0-255)**
0	34.037	0.975	0.022	5.696
0.01	34.033	0.973	0.022	5.635
0.1	33.783	0.961	0.023	5.777
1	30.585	0.825	0.031	8.003
10	20.764	0.375	0.097	24.802
100	16.314	0.526	0.169	43.032
$\alpha = 100, \gamma = 1000, \delta = 10$, train_epoch=300				

Table 4.8 The ICVL-RMSE changes against the β variations

β	**PSNR**	**SSIM**	**RMSE (0-1)**	**RMSE (0-255)**
0	44.432	0.998	0.007	1.794
0.01	36.727	0.884	0.015	3.821
0.1	24.098	0.446	0.070	17.954
1	24.047	0.826	0.072	18.260
10	24.274	0.784	0.075	19.239
100	24.274	0.808	0.075	19.116
$\alpha = 100, \gamma = 1000, \delta = 10$, train_epoch=300				

For the above phenomenon, we explain that KL-divergence tunes the encoder by forcing the latent vector of z generated to be close to the normal distribution. The learning gradients only pass through the encoder. Only the encoder has to be updated. However, GAN adjusts the whole generator network, including the encoder and decoder, by making the outputs more like the real ones. The gradients go through the whole generator. The decoder and encoder have to be updated together. So, the GAN has higher efficiency to train the generator.

However, the above explanation is only our hypothesis; it lacks a strong theory and experimental supports. We need to undertake further research to address this problem soon.

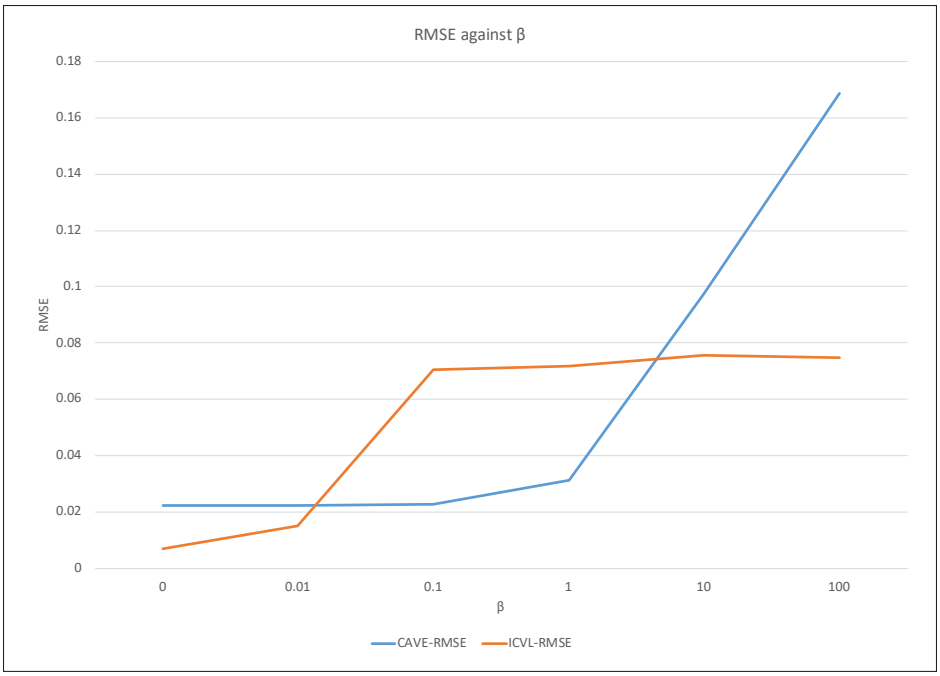

Figure 4.12 The RMSE against the β

4.7 Conclusion and future work

In this paper, we first introduced the challenge of MSIs' reconstruction from RGBs and the related works' common shortages. Because their approaches are based on static and dependent neural networks, they cannot generate new variations to supplement the lost information between the MSIs and RGBs. Next, we analyzed the bottle neck of the problem and elaborated on our proposed approach, which leverages re-parameterizing latent vectors and GAN tricks to create new variational MSI-like images. In this way, one latent vector can evolve into many latent vectors respecting the normal distribution. One input RGB image with random latent space vectors can be created out of the lost possible multiple outputs. We then used the GAN and L1 regulator to make the possible multiple outputs convergence into one real-like MS image output. Thus, we were able successfully to solve the metamerism problem and transform the one-to-many problem into a one-to-one problem by bringing in random latent vectors. We used qualitative and quantitative methods to evaluate our approach to the CAVE and ICVL datasets.

With much less training data than the previous approaches, we got comparable results on the CAVE dataset and surpassed the state-of-the-art results on the ICVL dataset.

In the future, we plan to conduct more research on optimizing the hyperparameter settings. We will expand our approach to more datasets to verify its versatility.

4.8 The supplementary

4.8.1 The detailed neural Network Architecture

In this section, we list all the detailed structures of the neural networks dealt with in Section 4.4 in Tables 4.9, 4.10, 4.11, and 4.12 respectively. The training batch size is 10240 and the testing batch size is 262144 (512×512). The training epoch is 300.

Table 4.9 The generator of RGB_to_MSI

	Layer	**Input**	**Output**	**Activation**
Encoder	Dense	RGB(3)	512	Leaky_Relu
	Dense	512	512	Leaky_Relu
	Dense	512	100+100	N/A
Z	Latent Re-parameter	100+100	100	N/A
Decoder	Dense	512	512	Leaky_Relu
	Dense	512	512	Leaky_Relu
	Dense	512	MSI(31)	Tanh

Table 4.10 The discriminator of RGB_to_MSI

	Layer	**Input**	**Output**	**Activation**
Discriminator	Dense	MSI(31)	64	Leaky_Relu
	Dense	64	64	Leaky_Relu
	Dense	64	1	Sigmoid

Table 4.11 The generator of MSI_to_RGB

	Layer	Input	Output	Activation
Encoder	Dense	MSI(31)	512	Leaky_Relu
	Dense	512	512	Leaky_Relu
	Dense	512	100+100	N/A
Z	Latent Re-parameter	100+100	100	N/A
Decoder	Dense	512	512	Leaky_Relu
	Dense	512	512	Leaky_Relu
	Dense	512	RGB(3)	Tanh

Table 4.12 The discriminator of MSI_to_RGB

	Layer	Input	Output	Activation
Discriminator	Dense	RGB(3)	64	Leaky_Relu
	Dense	64	64	Leaky_Relu
	Dense	64	1	Sigmoid

CHAPTER 5

TAIJIGNN: A NEW CYCLE-CONSISTENT GENERATIVE NEURAL NETWORK FOR HIGH-QUALITY BIDIRECTIONAL TRANSFORMATION BETWEEN RGB AND MULTISPECTRAL DOMAINS A NEW CYCLE-CONSISTENT GENERATIVE NEURAL NETWORK FOR HIGH-QUALITY BIDIRECTIONAL TRANSFORMATION

Xu Liu[1] , Abdelouahed Gherbi[1] , Wubin Li[2] , Zhenzhou Wei[3] , Mohamed Cheriet[1]

[1] Synchromedia Laboratory, Université du Québec (École de technologie supérieure),
1100 Notre-Dame Ouest, Montréal, Québec, Canada H3C 1K3
[2] Ericsson Canada,
8275 Trans Canada Route, Saint-Laurent, Québec, Canada H4S 0B6
[3] Department of Electrical and Computer Engineering, McGill University,
845 Sherbrooke Street West, Montréal, Québec, Canada H3A 0G4

Article published on the journal « Sensors » in August 2021,
DOI: 10.3390/s21165394

5.1 Abstract

Since multispectral images (MSIs) and RGB images (RGBs) have significantly different definitions and severely imbalanced information entropies, the spectrum transformation between them, especially reconstructing MSIs from RGBs, is a big challenge. We propose a new approach, the Taiji Generative Neural Network (TaijiGNN), to address the above-mentioned problems. TaijiGNN consists of two generators, G_MSI, and G_RGB. These two generators establish two cycles by connecting one generator's output with the other's input. One cycle translates the RGBs into the MSIs and converts the MSIs back to the RGBs. The other cycle does the reverse. The cycles can turn the problem of comparing two different domain images into comparing the same domain images. In the same domain, there are neither different domain definition problems nor severely underconstrained challenges, such as reconstructing MSIs from RGBs. Moreover, according to several investigations and validations, we effectively designed a multilayer perceptron neural network (MLP) to substitute the convolutional neural network (CNN) when implementing the generators to make them simple and high performance.

Furthermore, we cut off the two traditional CycleGAN's identity losses to fit the spectral image translation. We also added two consistent losses of comparing paired images to improve the two generators' training effectiveness. In addition, during the training process, similar to the ancient Chinese philosophy Taiji's polarity Yang and polarity Yin, the two generators update their neural network parameters by interacting with and complementing each other until they all converge and the system reaches a dynamic balance. Furthermore, several qualitative and quantitative experiments were conducted on the two classical datasets, CAVE and ICVL, to evaluate the performance of our proposed approach. Promising results were obtained with a well-designed simplistic MLP requiring a minimal amount of training data. Specifically, in the CAVE dataset, to achieve comparable state-of-the-art results, we only need half of the dataset for training; for the ICVL dataset, we used only one-fifth of the dataset to train the model, but obtained state-of-the-art results.

5.2 Introduction

5.2.1 The background

Different wavelength lights have various propagating, reflecting, and refracting properties. Multispectral images (MSIs) use distinct bands to record different wavelength properties and contain affluent spectral information Wikipedia (2020d). With the rich spectral information, MSIs can help to solve problems that RGBs can not, such as MSIs for the retinal oximetry measurements in four diseased eyes Dwight, Weng, Coffee, Pawlowski & Tkaczyk (2016), the detection of adulteration with different grains in wheat products Verdú *et al.* (2016), the non-contact analysis of forensic traces Edelman, Gaston, van Leeuwen, Cullen & Aalders (2012), and the visualization of latent traces in crime scenes Edelman & Aalders (2018).

Although MSIs have a broad application scope, acquiring them is a time-wasting, complicated and expensive process. Tens or hundreds of MSI bands have to be taken one by one, which consumes a significant amount of time and storage space resulting in big, complicated, and expensive multispectral apparatuses.

On the other hand, RGB images (RGBs) have only three channels: blue, green, and red. Additionally, they form other colors by composing the three primary colors with different ratios, and their formation principle is different from that for spectral images. Moreover, RGBs can be quickly and cheaply obtained by ordinary consumer cameras, such as mobile phone cameras. Although the RGB images can give us a good color impression about the objects or the scenarios, they contain scarce spectral information and have many restrictions.

In sum, RGBs and MSIs have their pros and cons. It would be meaningful to create a model that can perform bi-directional translations between RGBs and MSIs. However, there are two significant challenges to be solved first.

The first challenge is that RGBs and MSIs have different definitions and formation mechanisms. Although we can synthesize RGBs from MSIs with color matching functions, they can work under limited circumstances, as long as there is sufficient visible light (380 nm–700 nm) spectral information. In other circumstances, when we only have several bands, such as near-infrared (NIR), infrared (IR), and ultraviolet (UV), it is difficult to use the color matching function to synthesize a real RGB-like image Sun, Jung, Fu & Han (2019). Moreover, the color matching functions are often too complex to accelerate in a parallel manner. Furthermore, no color matching function can synthesize MSIs from RGBs directly Parkkinen, Jaaskelainen & Kuittinen (1988).

The second challenge is that the spectral resolution of MSIs is much higher than that of the RGBs'. MSIs often contain much more information than RGBs. One RGB composition may map to many kinds of MSI compositio,n named metamerism Chen & Liu (2014). Reconstructing MSIs from RGBs is a severely underconstrained problem.

Most previous related works meet a bottleneck when trying to improve reconstructing MSI performance. As they treat the two reconstruction processes, RGBs->MSIs (converting RGBs to MSIs) and MSIs->RGBs (converting MSIs to RGBs) as two separate processes without any relations. They only concentrated on rebuilding MSIs from RGBs and ignored the process of MSIs->RGBs, since they thought that MSIs->RGBs is obvious and no help to reconstruct

MSIs from RGBs. Thus, they did not discover the link between the two processes. However, the two processes have a complementary relation. If we treat them separately, we may lose a large amount of valuable information, which would lead to poor reconstruction and miss the opportunity to solve the above two problems.

Our approach TaijGNN is inspired by the ancient Chinese philosophy "Taiji" Wikipedia (2020c) and Cycle Generative Adversarial Network (CycleGAN) Zhu *et al.* (2017) and merged with some of our recent considerations. It has two in-out ports and two generators. One port is for inputting or outputting RGBs. The other port is for MSIs. Generator G_{MSI} converts the RGBs into MSIs. Generator G_{RGB} converts the MSIs into RGBs. Moreover, the two ports and two generators are formed into two cycles, (5.1) and (5.2).

$$\underset{(Original)}{RGB} \overset{G_{MSI}}{\rightarrow} \underset{(Generated)}{MSI'} \overset{G_{RGB}}{\rightarrow} \underset{(CycleGenerated)}{RGB"} \tag{5.1}$$

$$\underset{(Original)}{MSI} \overset{G_{RGB}}{\rightarrow} \underset{(Generated)}{RGB'} \overset{G_{MSI}}{\rightarrow} \underset{(CycleGenerated)}{MSI"} \tag{5.2}$$

Since the most important fact causing the above two challenges is that RGBs and MSIs belong to two distinct domains, the conversions between them would be similar to the conversions between images and labels. The two cycles, (5.1) and (5.2), ingeniously covert the problem of two different domain conversion into the same domain conversion. As we know, it is not difficult to convert one image to itself in the same domain. In this way, the two challenges can be solved naturally.

Moreover, most researchers believe that complex deep neural networks would improve the models' performance, and the more complex the neural networks are, the better effect they will have. Therefore, their approaches have often adopted tens or hundreds of convolutional neural network (CNN) layers with hundreds of residual network layers, which leads to a long computing time and high power consumption Perera, Abavisani & Patel (2017); Stiebel, Koppers, Seltsam & Merhof (2018); Can & Timofte (2018). According to our thorough investigations and experiments, normal 2D convolutional operations scarcely improve the spectrum reconstruction performance and make

the generated images blurry. Therefore, in our generators, we remove the traditional convolution layers and mainly adopt multilayer perceptron (MLP) architecture. The following experiments (Section 4.5) prove that TaijiGNN based on MLP has a higher performance than the related works based on CNN.

Additionally, unlike traditional CycleGAN, in TaijiGNN, we also compare the generated image with the paired ground truth image to enhance the generator's training performance. For example, in Cycle (5.1), when we used Generator G_{MSI} to convert the RGB into the MSI, we could calculate their mean absolute error (MAE) to update Generator G_{MSI}. On the contrary, the traditional CycleGAN can only use adversary networks to determine if the generated images belong to the domain due to lacking the paired dataset. Moreover, we found that the indirect adversary network learning efficiency is much lower than that of direct supervised learning. Therefore, TaijiGNN eliminates the adversary network in its architecture, dramatically increasing the training speed and the model's performance.

Figure 5.1 shows TaijiGNN's reconstruction performance. We used the trained model to reconstruct an MSI from an RGB and integrated eight selected bands into one image. Additionally, the reconstruction error map presents the degree of dissimilarity. More details are provided in the following sections.

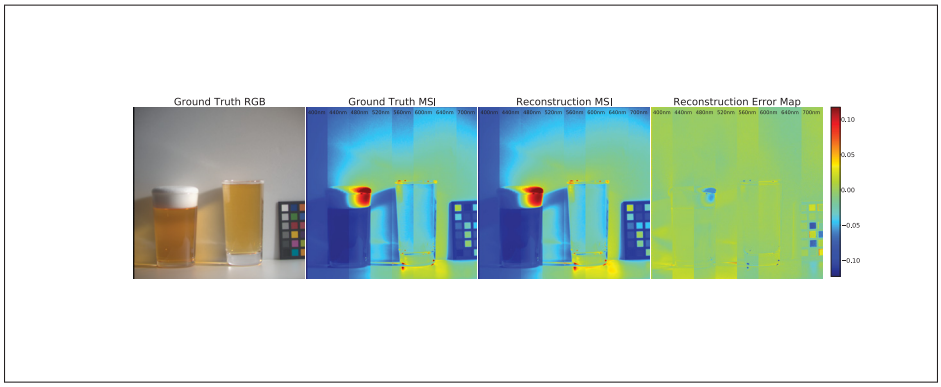

Figure 5.1 An example of reconstructing a multispectral image from an RGB image

5.2.2 The Contributions

Our contributions are mainly in the following three aspects:

- We proposed a new neural network, TaijiGNN (Taiji Generative Neural Network), which takes advantage of cycled neural network architecture to convert the problem of comparing two different domain images into comparing the same domain images, which can supply a solid theory foundation to solve the two distinct domain image translation problems.
- We effectively designed a multilayer perceptron neural network to substitute the convolutional neural network when implementing the generators. Additionally, we used solid experiments to prove that our MLP generators perform better than the traditional generators based on CNN during spectral super-resolution.
- We innovatively eliminated the traditional CycleGAN's identity loss to tackle the different domain images with different dimensions and induced two consistency losses in TaijiGNN to enhance the training performance.

The remaining parts of the paper are organized as follows: First, we demonstrate several related studies in Section 5.3. Then, the proposed approach is elaborated in Section 5.4. Next, in Section 4.5, the qualitative and quantitative experiments and comparisons conducted to assess our approach are described. Moreover, we discuss the proposed approach's limitations in Section 4.6. Furthermore, in the last Section 5.7, a summary and future work are introduced. In Appendix 4.8, we provide the details of TaijiGNN's training and inferencing performance information.

5.3 Related work

As described in Section 5.2, direct spectrum translations between MSIs and RGBs have many benefits. Many researchers have contributed to this area.

In 2014, Nguyen et al. combined whited balanced RGB images and the radial basis function (RBF) network to reconstruct multispectral images. The performance of their approach is acceptable when the spectrum of illumination and reflectance is flat. Nevertheless, when the

spectrum becomes spiked, their method failed to maintain a good performance. Their work also has some other restrictions, such as that all scenes are under the same illumination Nguyen *et al.* (2014).

Two years later, in 2016, with the RGB projection and the sparse dictionary of multispectral signatures, Arad et al. successfully improved the conversion of MSIs from RGBs. They obtained top results at that time. However, their method must sample some information from every RGB image and MSI to make the multispectral image prior to reconstructing the MSIs, narrowing their applied fields. Additionally, their MSI recovery quality heavily depends on the scenes Arad & Ben-Shahar (2016).

The previous methods were overly reliant on all kinds of prerequisites, such as the environmental factors or the device's specifications. Once the prerequisites change, they have to rebuild their models according to the new prerequisites. On the contrary, deep learning approaches are data-driven. Once we finish the deep learning network architecture design, it is not necessary to redesign the whole network even when the scenes change. Re-training the model with the new dataset is the only necessary action. Therefore, an increasing number of spectrum conversion approaches take advantage of this recent technology.

Choi et al. leveraged an autoencoder to restore MSIs from RGBs in 2017. Their approach avoided the trade-off between spectral accuracy and spatial resolution. However, it only involves restoring hyperspectral images from compressive spectral images, but not reconstructing the MSIs from the RGBs Choi, Jeon, Nam, Gutierrez & Kim (2017). In the same year, Zhiwei et al. suggested a unified deep learning framework named HSCNN, reconstructing MSIs from RGBs. They used simple interpolation to upsample RGBs and leveraged the paired upsampled RGBs and MSIs to train the CNN model. Furthermore, they declared that their approach surpassed all previous approaches Xiong *et al.* (2017).

In 2018, Zhan et al. upgraded HSCNN's RGB upsampling process from a handcrafted operation to a simple CNN. They named their new approach "HSCNN+". Additionally, they replaced the HSCNN's regular convolution layers with residual blocks or dense blocks. Additionally, they

gave the name “HSCNN-R” to the network with residual blocks and named the network with the dense blocks “HSCNN-D” Shi *et al.* (2018). Moreover, HSCNN-D was the winner, and HSCNN-R achieved second place in the NTIRE 2018 Spectral Reconstruction Challenge Arad *et al.* (2018).

In the same year, 2018, Berk et al. successfully produced a multispectral image from an RGB image in the wild. They first assessed the sensitivity function from an RGB image, and then proposed a classifier network to predict which function can be used to constitute the image. Finally, they built a reconstruction model to convert the RGB image to the multispectral image Kaya *et al.* (2018). They obtained the seventh place in the NTIRE 2018 Spectral Reconstruction Challenge, and they declared that their approach has the fewest layers and the fastest speed.

Moreover, Kin et al. demonstrated how to leverage the conditional GAN to conduct bi-directional spectrum translation between multispectral images and RGB images in the 2019 Conference on Computer Vision and Pattern Recognition. Although they made some progress in the spectrum reconstruction performance, they did not give a reasonable explanation of how to supplement the lost multispectral information Gwn Lore *et al.* (2019).

Recently, in 2020, in the “NTIRE Challenge on Spectral Reconstruction from an RGB Image”, several new approaches emerged Arad *et al.* (2020). The champion, Jiaojiao et al., suggested using the adaptive weighted attention network to restore the multispectral image from the RGB image. Their neural network mainly consists of several dual-residual attention blocks. Additionally, they obtained the best result in the “Clean” dataset and ranked third in the “Real World” dataset Li *et al.* (2020).

Although the previous related papers have achieved some progress in reconstructing spectral images, no one clearly explained how their approaches solve the underconstrained spectrum translation in theory. Moreover, most related works neglect the differences between spectrum reconstruction problems and other image processing, try to use traditional stacking and design complex CNN layers to solve the problem. However, since traditional CNNs mainly perform

convolutions on the 2D surface, except addition operations, there are no convolutional operations on the spectral dimension. The relation among different spectral bands may be overlooked. Additionally, deep neural networks may improve the generator's performance, but too many neural layers may cause a series of gradient vanishing, overfitting, and long executing time problems. Furthermore, most of their works only focus on reconstructing MSIs from RGBs, but ignore the process of reconstructing RGBs from MSIs. In fact, the two directions, from MSIs to RGBs and the reverse, have complementary relations. If we could take advantage of these relations, we could improve the spectrum reconstruction. In the following sections, we try to solve the above problems with our approach.

5.4 The proposed method

5.4.1 The problem analysis

Our target is to build a model to efficiently perform bidirectional translation between RGBs and MSIs. To achieve this target, we need to solve two significant challenges. One challenge is that RGBs and MSIs belong to two specific domains with different definitions and synthesis rules. There is no direct mapping relation between the two domains. The other is how to reconstruct the multispectral information, which does not exist in the RGB image. Multispectral images have more abundant information than the RGBs.

Figure 5.2 shows the CIE 1931 RGB color matching function Wikia (2021), which clearly shows the relation between the RGBs and the MSIs. Any color pixel can be created by mixing three primary colors—-red, green, and blue. On the contrary, multispectral pixels comprise different wavelength light bands. Additionally, each band only records the luminosity distribution of the specific wavelength light. Therefore, MSIs and RGBs have fundamentally different definitions and formats. There is no direct mapping relation between them.

Additionally, Figure 5.3 demonstrates the second challenge. RGBs only have three bands. However, MSIs have tens or hundreds of bands. Thus, the possible space of MSIs is much larger

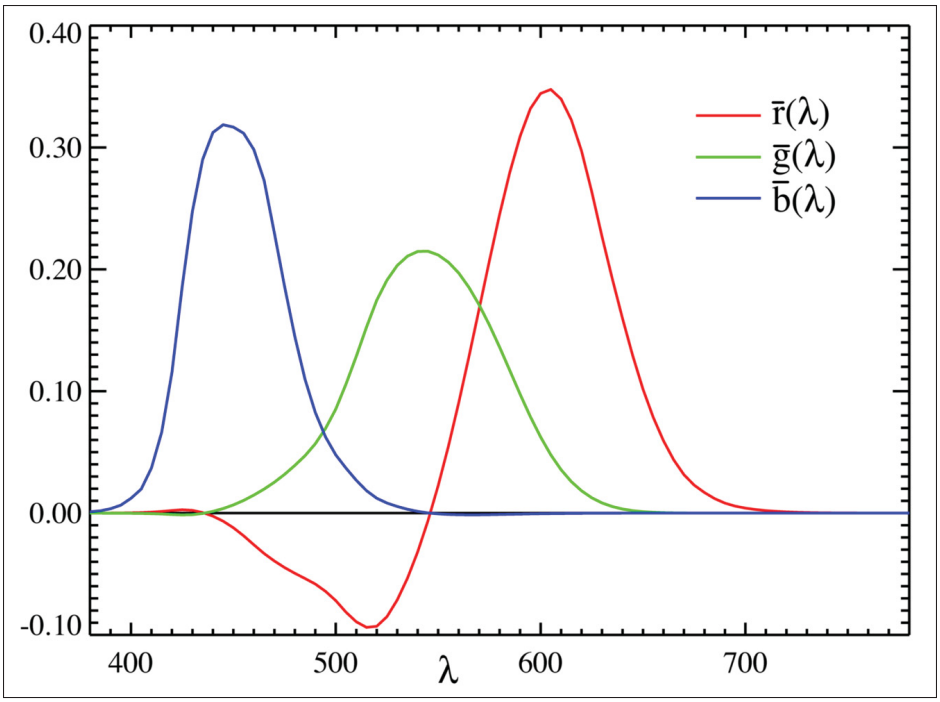

Figure 5.2 The CIE 1931 RGB color matching function
Taken from wikia (2021, Website)

than the possible space of RGBs. There is a massive information entropy gap between the two domains. The multispectral images contain much more information that does not exist in the RGB images. If we use RGBs to represent MSIs, we will meet the "metamerism" problem, which means one R, G, and B band combination may map to many MSI band combinations. The metamerism is an extremely underconstrained problem and will make the model's training hard to reach convergence.

5.4.2 The proposed approach

To solve the above two problems, we proposed TaijiGNN. Since this idea is mainly inspired by the cycle generative adversarial network (CycleGAN) and the ancient Chinese philosophy Taiji's concept, we first introduce and compare them. Table 5.1 shows the symbols used in Figure 5.4.

Figure 5.4a is the original CycleGAN's schematic diagram, which has two cycles, $x \xrightarrow{G} \hat{Y} \xrightarrow{F} \hat{x}$ and $y \xrightarrow{F} \hat{X} \xrightarrow{G} \hat{y}$. Generator G translates sample x of the domain X into sample $\hat{Y}$ of the domain

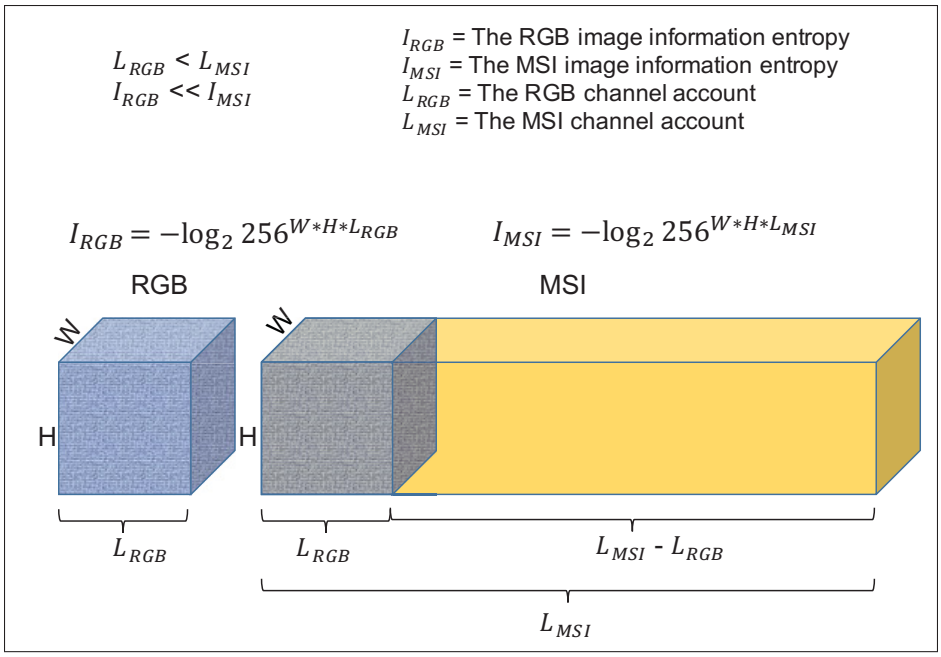

Figure 5.3 The difference of RGB and MSI image information entropies

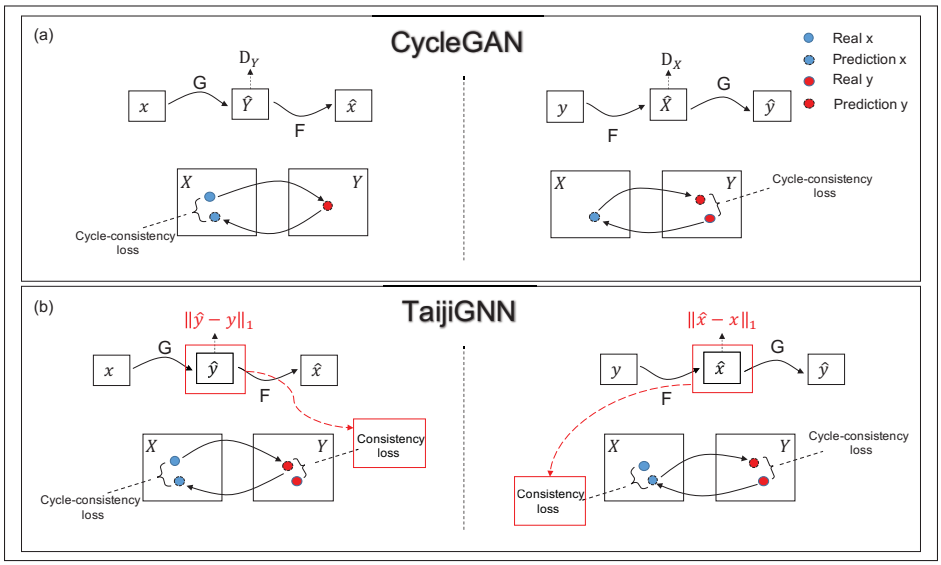

Figure 5.4 (a) The traditional CycleGAN's schematic diagram taken from Google (2020, Website) (b) The TaijiGNN's schematic diagram

Table 5.1 The symbols table of Figure 5.4

Symbol	Meaning
x	Original sample from X domain
y	Original sample from Y domain
$\hat{x}$	Sample generated by Generator F and paired with y
$\hat{y}$	Sample generated by Generator G and paired with x
X	Domain X
Y	Domain Y
$\hat{X}$	Sample generated by Generator F and unpaired with y
$\hat{Y}$	Sample generated by Generator G and unpaired with x
G	Generator, translate samples from domain X into domain Y
F	Generator, translate samples from domain Y into domain X
D_X	Discriminator of domain X
D_Y	Discriminator of domain Y

Y. In the flow $x \xrightarrow{G} \hat{Y} \xrightarrow{F} \hat{x}$, we use $\hat{Y}$ instead of $\hat{y}$ since there are no paired datasets in CycleGAN; the generated sample belongs to domain Y, but is not specified to one and has no direct mapping relation with the original x. Then, we translate the $\hat{Y}$ into $\hat{x}$ with Generator F. Here, we aim for the $\hat{x}$ to be as close to x as possible, which can be used to train Generators F and G. Moreover, we use the original Y and the generated $\hat{Y}$ to train Discriminator D_Y and Generator G. A similar occurred in the backward cycle $y \xrightarrow{F} \hat{X} \xrightarrow{G} \hat{y}$.

The most significant advantage of the original CycleGAN is that it can translate unpaired images by the cycle structure between two different domains, such as the horse and zebra Zhu *et al.* (2017). However, in our case, we have paired RGBs and MSIs, which is a great advantage. Due to this advantage, we designed a novel supervised cycle-consistent generative neural network, "TaijiGNN".

Figure 5.4b shows TaijiGNN's schematic diagram, which also has two cycles, forward, $x \xrightarrow{G} \hat{y} \xrightarrow{F} \hat{x}$ and backward, $y \xrightarrow{F} \hat{x} \xrightarrow{G} \hat{y}$. The most significant difference highlighted in red shows that when we use G to translate the x into samples of domain Y, $\hat{y}$ replaces the $\hat{Y}$. Since paired datasets are available in this case, we can use L_1 loss to enhance the G's training performance. A similar situation occurs in the backward cycle $y \xrightarrow{F} \hat{x} \xrightarrow{G} \hat{y}$. Moreover, we omit the two

discriminators D_X and D_Y, as in our case, we have the paired dataset, L_1 loss is enough to train a good model in the TaijiGNN. Extra adversary networks will bring in more noise and reduce the model's performance.

Moreover, we anticipate that TaijiGNN will have better performance than the original CycleGAN since TaijiGNN has extra direct paired image consistency loss information, enhancing the generator's training efficiency. Therefore, each generator of TaijiGNN can learn from the cycle-consistency loss of the source domain data and the consistency loss of the target domain data simultaneously.

Although the MSI domain and the RGB domain have different definitions and possible space sizes, its cycle architecture of TaijiGNN successfully converts comparing different domain images into comparing the same domain images. In the same domain, there are no domain matching and information supplement problems.

Furthermore, we named our neural network "Taiji" as it was mainly inspired by the ancient Chinese philosophy term "Taiji". Taiji consists of two polarities, "Yang" ("Positive") and "Yin" ("Negative"). Additionally, the "Yin" and "Yang" like atoms constitute every system in the universe. Within each "Yin" and "Yang" system, the "Yin" and "Yang" have opposite properties and continue to move and change all the time. The most important factor is the two polarities do not have strict boundaries and can be converted to each other under some circumstances. In the system, the "Yin" and "Yang" give rise to and complement each other to make the system reach a dynamic balance Wikipedia (2020c). We think Taiji's characteristics and working mechanism are consistent with our cycle-consistent generative neural network. The two generators act as Taiji's two polarities, and during training, the two generators compose a whole system and help each other converge. Therefore, we name our neural network "TaijiGNN".

5.4.3 The detailed implementation of the approach

5.4.3.1 The model architecture

Figure 5.5 is the detailed implementation of TaijiGNN. It mainly consists of two generators, G or $Generator_{RGB}$ (converting the MSI to the RGB) and F or $Generator_{MSI}$ (converting the RGB to the MSI). One generator's output is connected with the other's input. From different directions, they form two cycles, Cycle (5.3) and Cycle (5.4).

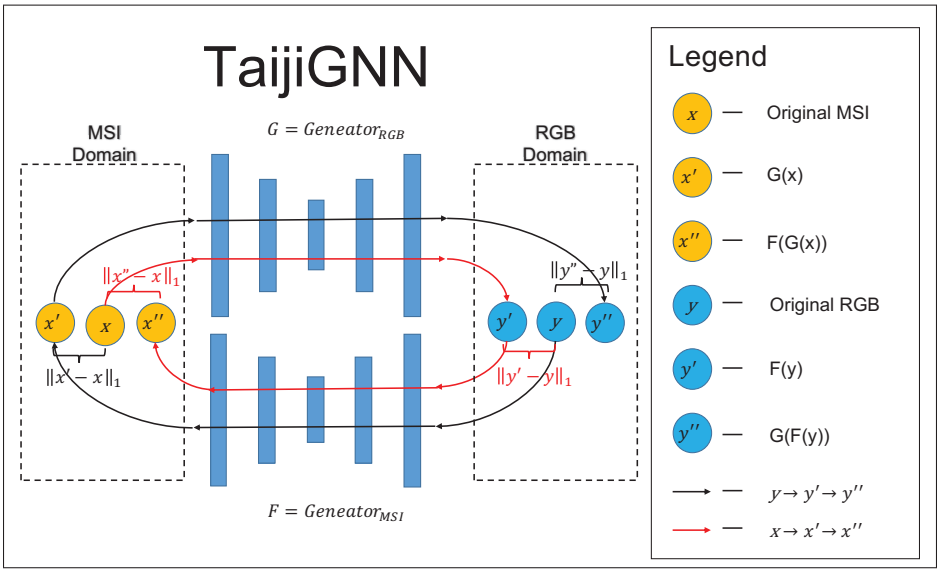

Figure 5.5 The detailed implementation of TaijiGNN

$$\underset{(Original)}{x} \xrightarrow{G} \underset{(Generated)}{y'} \xrightarrow{F} \underset{(CycleGenerated)}{x''} \tag{5.3}$$

$$\underset{(Original)}{y} \xrightarrow{F} \underset{(Generated)}{x'} \xrightarrow{G} \underset{(CycleGenerated)}{y''} \tag{5.4}$$

The significant difference between the standard CycleGAN and TaijiGNN is that TaijiGNN has two extra L_1 consistency losses, Loss $\mathcal{L}_{RGB}$ (5.5) and Loss $\mathcal{L}_{MSI}$ (5.6). These two losses can

help to improve the two generators' performance by calculating the mean absolute error (MAE) between the ground truth image and the generated image.

The consistency loss from MSIs to RGBs:

$$\mathcal{L}_{RGB} = \|G(x) - y\|_1 = \|y' - y\|_1 \tag{5.5}$$

The consistency loss from RGBs to MSIs:

$$\mathcal{L}_{MSI} = \|F(y) - x\|_1 = \|x' - x\|_1 \tag{5.6}$$

Additionally, we keep the two cycle consistency losses, Loss $\mathcal{L}_{cycle\_MSI}$ (5.7) and Loss $\mathcal{L}_{cycle\_RGB}$ (5.8) in our approach. Loss $\mathcal{L}_{cycle\_MSI}$ guarantees that the cycled generated image MSI" is as close as the ground truth MSI when finishing Cycle (5.3). Similarly, Loss $\mathcal{L}_{cycle\_RGB}$ ensures that the cycled generated image RGB" is as close as the ground truth RGB in Cycle (5.4).

The cycle consistency loss from MSIs via RGBs to MSIs:

$$\mathcal{L}_{cycle\_MSI} = \|F(G(x)) - x\|_1 = \|F(y') - x\|_1 = \|x'' - x\|_1 \tag{5.7}$$

The cycle consistency loss from RGBs via MSIs to RGBs:

$$\mathcal{L}_{cycle\_RGB} = \|G(F(y)) - y\|_1 = \|G(x') - y\|_1 = \|y'' - y\|_1 \tag{5.8}$$

Moreover, unlike the normal CycleGAN, TaijiGNN eliminates identity loss. In the normal CycleGAN, the input and output have identical dimensions; however, the RGB and the MSI have different dimensions in our scenario. It is neither meaningful nor possible to compare two images with different dimensions.

We also cut off two discriminator losses of the traditional CycleGAN. The reasons for this are as follows: the traditional CycleGAN does not have a paired dataset, so it must use the discriminator loss to help to train the generator. However, in our scenario, since we have a paired dataset, some L1 losses, such as MAE, have a higher performance than the discriminator loss. Additionally, training through discriminator losses is an indirect training method, which will make the generated image blurry. Overall, direct training is better than indirect training.

The two generators of TaijiGNN take the core roles of converting images between the MSI domain and the RGB domain. The choice of architecture directly determines the performance of the system. We tried several mainstream architectures, including the convolutional neural netwok (CNN), residual neural network, U-Net and Multi-Layer Perception (MLP). The MLP has the best performance.

Regarding the above phenomenon, our explanation is as follows: In our scenario, the main target is to produce the spectral super-resolution, not the spatial super-resolution. The traditional neural networks based on CNN can effectively extract information distributed on the two spatial dimensions, such as shape. However, the required spectral information is distributed on the third dimension. The neural networks based on 1-D MLP focus on extracting spectral features, neglecting the 2-D spatial information. Therefore, 1-D MLPs often obtain better results than 2-D CNNs.

Based on the above analysis and our real verifications, TaijiGNN's two generators adopt MLP architecture. The detailed network architectures are shown in Tables 4.9 and 4.11, respectively.

Table 5.2 The architecture of Generator F_{MSI}

	Layer	**Input**	**Output**	**Activation Function**
F_{MSI}	Dense	RGB(3)	512	Leaky Relu
	Dense	512	512	Leaky Relu
	Dense	512	MSI(31)	tanh

Table 5.3 The architecture of Generator G_{RGB}

	Layer	Input	Output	Activation Function
G_{RGB}	Dense	MSI(31)	512	Leaky Relu
	Dense	512	512	Leaky Relu
	Dense	512	RGB(3)	tanh

5.4.3.2 The model training

The core training goal is to make the two generators, G_{RGB} and F_{MSI}, converge. The primary approach is to apply gradient descent on the previous four losses, $\mathcal{L}_{RGB}$ (5.5), $\mathcal{L}_{MSI}$ (5.6), $\mathcal{L}_{cycle\_MSI}$ (5.7), and $\mathcal{L}_{cycle\_RGB}$ (5.8).

Figure 5.6 demonstrates the whole training flow of TaijiGNN. From Figure 5.6, we can find that the whole training flow consists of two symmetrical cycles, Cycle $x(MSI) \xrightarrow{G} y'(RGB') \xrightarrow{F} x''(MSI'')$ marked with red solid and dash lines, and Cycle $y(RGB) \xrightarrow{F} x'(MSI') \xrightarrow{G} y''(RGB'')$ highlighted with black solid and dash lines. Since they are symmetrical, we only describe Cycle $y(RGB) \xrightarrow{F} x'(MSI') \xrightarrow{G} y''(RGB'')$ in detail.

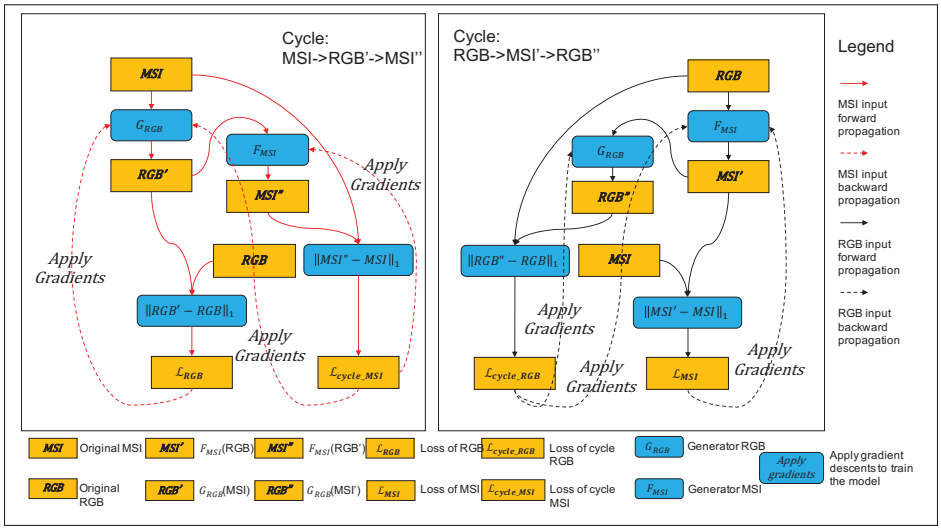

Figure 5.6 The detailed training flow of TaijiGNN

Cycle $y(RGB) \xrightarrow{F} x'(MSI') \xrightarrow{G} y''(RGB'')$ starts from inputting an RGB image into Generator F_{MSI}, which obtains a reconstructed MSI image MSI'. After this operation, the flow splits

into two branches. One branch calculates $\mathcal{L}_{MSI}$ by computing the MAE between the original MSI and the generated MSI'. Additionally, when we obtain Loss $\mathcal{L}_{MSI}$, we can apply gradient descents to update Generator F_{MSI}. The other branch inputs the MSI' into Generator G_{RGB} to obtain the cycled generated RGB'' and then to calculate Loss $\mathcal{L}_{cycle\_RGB}$ by comparing the original RGB with the cycled generated RGB''. Once we obtain Loss $\mathcal{L}_{cycle\_RGB}$, we can apply gradient descents to update Generator F_{MSI}, and Generator G_{RGB}. A similar training flow occurs in Cycle $y(RGB) \xrightarrow{F} x'(MSI') \xrightarrow{G} y''(RGB'')$.

Figure 5.6 demonstrates a classical TaijiGNN training epoch. In each training epoch, Generator G_{RGB} and Generator F_{MSI} are updated three times, respectively. In detail, Generator G_{RGB} is updated by applying gradients, Loss $\mathcal{L}_{RGB}$, Loss $\mathcal{L}_{cycle\_MSI}$ and Loss $\mathcal{L}_{cycle\_RGB}$, respectively. Similarly, Generator F_{MSI} is updated by applying gradients, Loss $\mathcal{L}_{MSI}$, Loss $\mathcal{L}_{cycle\_RGB}$ and Loss $\mathcal{L}_{cycle\_MSI}$. Therefore, after several epochs of training, Generator G_{RGB} and Generator F_{MSI} eventually converge.

5.5 Experiments

The CAVE (Yasuma *et al.*, 2008) and ICVL (Arad & Ben-Shahar, 2016) are the two most classical spectral reconstruction experiment datasets. To compare our work with state-of-the-art works equally and evaluate our approach's performance thoroughly, we also selected CAVE and ICVL as our experiment datasets. We demonstrate the detailed experiment process and outcomes in the coming subsections.

5.5.1 The experiment requisites

We list all the hardware devices and software involved in Tables 5.4 and 5.5, respectively. And this verification environment is similar to our previous work (Liu *et al.*, 2021b).

Since the Graphics Processing Unit (GPU) plays a vital role in the training and the inferencing, we provide a detailed introduction. Our GPU is Nvidia TITAN Xp, which has 3840 parallel computing cores with a 1582 MHz frequency in our experiment. The GPU memory has 12 GB

GDDR5, 11.4 Gbps speed, and 547.7 GB/s bandwidth. In this experiment, since the image processing contains many high-density computing jobs, we leveraged the GPU to accelerate the training and inferencing process. Moreover, in Appendices 4.8, we provide a summary of TaijiGNN's performance on the GPU and the CPU.

Table 5.4 The hardware list

Hardware	**Models**
Central Processing Unit	Xeon(Intel) E5-1620 (3.5GHz)
Graphics Processing Unit	TITAN-Xp (NVIDIA)
RAM	DDR4 32GB (Samsung)
Mechanical Hard Disk Drive	Westwood 4TB
SSD	512GB (Intel)

Table 5.5 The software list

Software	**Version**
Operating system	Ubuntu 18.04.5 LTS
Programming language	Python 3.6.9
Machine learning framework	Tensorflow with GPU 2.3.1
GPU programming frameworkd	Nvidia CUDA 11.2.0

5.5.2 The CAVE dataset

5.5.2.1 The dataset processing and the hyperparameters setting

The CAVE dataset was created by Columbia University Computer Vision Laboratory. It consists of 32 scenes taken indoor, and the scenes are divided into five sections: real and fake, stuff, paints, skin and hair, and food and drinks. Each scene is constituted of a multispectral image with 31 bands and an RGB image with 3 bands. The MSI's 31 bands span from 400 nm to 700 nm visible light at 10 nm intervals. Furthermore, each image's spatial resolution is 512×512 pixels (Yasuma *et al.*, 2008).

For easy processing and comparison, we first converted the original PNG image dataset into the NumPy array, which contains 32 rows (scenes), 512×512 (spatial resolution), and 34

columns (bands or channels). The 0–30 columns match the 400 nm, 410 nm, . . . , 700 nm MSI channels. Simultaneously, the 31–33 columns map to the Red, Green, and Blue channels. Moreover, according to the scene's row number, we separated the dataset into two equal-sized parts. Images with even indexes, 0, 2, 4, . . . , 30 were placed into one group, whereas images, 1, 3, 5, . . . , 31, with odd indexes, were placed into another group. We rotated to use the odd group and the even group to train and test the model. Generally speaking, the amount of training data is proportional to the performance of the model. Figure 5.7 demonstrates this phenomenon (Ng, 2017). To achieve better performance, the training and testing ratio is often set as 80%:20% or 70%:30%. However, increasing the ratio of the training data unavoidably leads to overfitting results and restricts application domains.

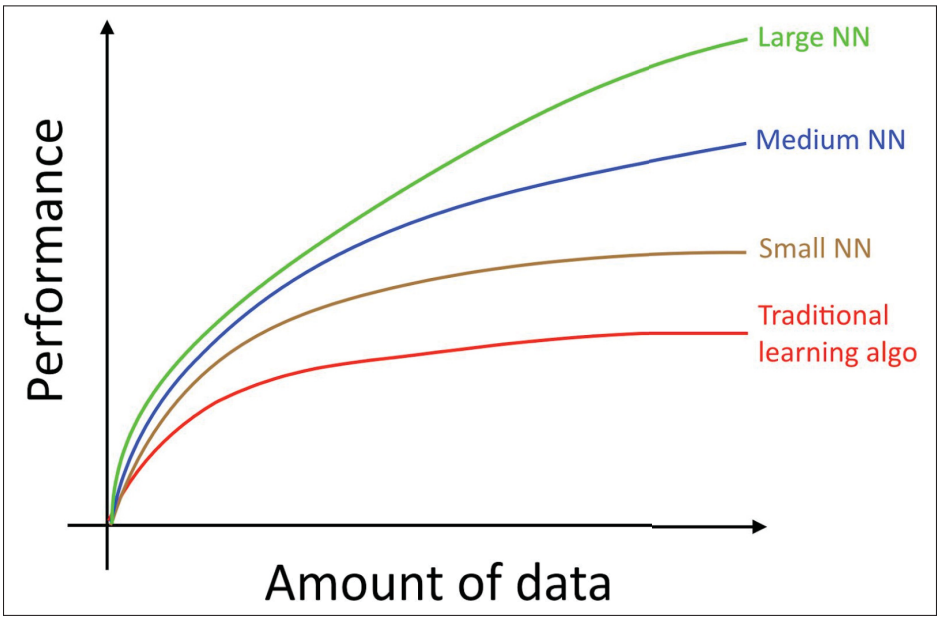

Figure 5.7 The relevance between the amount of data and the model's performance
Taken from Ng et al. (2021, Website)

We chose a more challenging method to split the CAVE, half for training, half for testing, which is the same as in the work of Kin et al. (Gwn Lore *et al.*, 2019). Moreover, Kin et al.'s conditional GAN is similar to our TaijiGNN. Both have generative network architecture. Therefore, their work was selected as the primary baseline. Furthermore, Arad et al. created the ICVL dataset

and declared that they achieved advantage outcomes at that time (Arad & Ben-Shahar, 2016). Berk et al. announced that they were the first to successfully reconstruct a multispectral image from an RGB image taken in unrestricted configuration (Kaya *et al.*, 2018). Therefore, Arad and Berk's works were also selected for comparison.

Moreover, we performed qualitative and quantitative evaluations of the bi-directional conversion between the RGB image and the multispectral image in the experiment.

Additionally, the detailed training parameter settings were as follows: There were 300 training epochs, and the training and the inferencing batch size was 10,240. We adopted the Adam optimizer, whose learning rate is 2×10^{-4}.

5.5.2.2 The qualitative evaluation

The qualitative evaluation was conducted in two stages: reconstructing multispectral images from RGB images and the reverse. Compared with related work introduced in Section 5.3, one advantage of TaijiGNN is that it can finish the above two spectral translations simultaneously.

Figure 5.8 is the qualitative effect of reconstructing the MSI from the RGB with Generator F_{MSI}. As in (Gwn Lore *et al.*, 2019), five spectral bands of image "beads" were chosen as samples. Additionally, the ground truth input RGB image was shown in the top-left image of Figure 5.9. The five selected bands span from 400 nm to 700 nm with approximate equal intervals. Since the naked eye cannot identify lights with such subtly different wavelengths, it becomes a prevalent and standard method to select several bands to present the qualitative spectral reconstruction results (Arad & Ben-Shahar, 2016; Kaya *et al.*, 2018). Moreover, in Section 5.5.2.3, we calculated the RMSE for the whole spectrum, from 400 nm to 700 nm, and drew the RMSE curve in Figure 5.10.

Additionally, we use the error map to present the degree of dissimilarity between the ground truth image and the reconstruction image. The error maps were generated by subtracting the original image from the forecast image. Additionally, we set the colormap equaling "jet" to pseudocolor

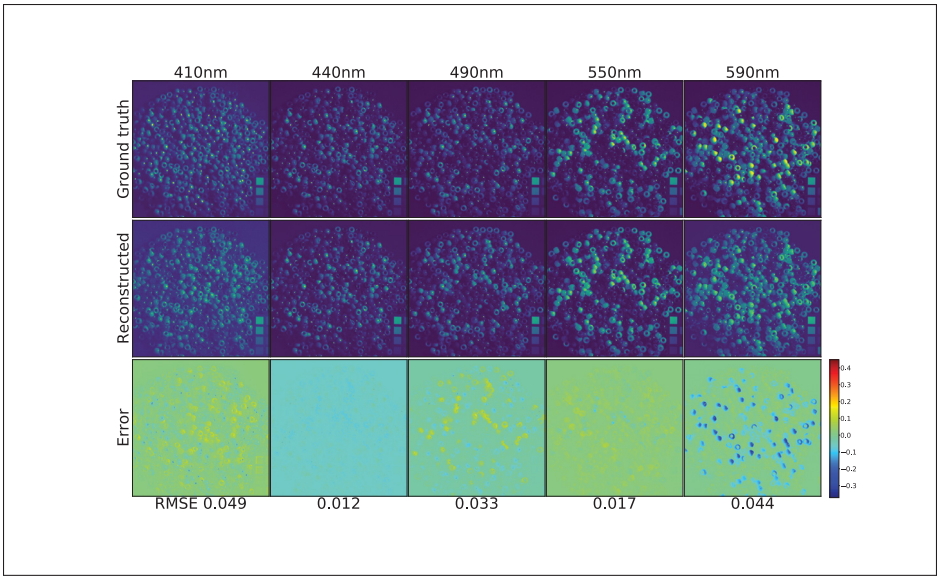

Figure 5.8 The effects of reconstructing the 5 chosen spectrum channels from an RGB image (The original input RGB is shown in at the top-left image of Figure 5.9.)

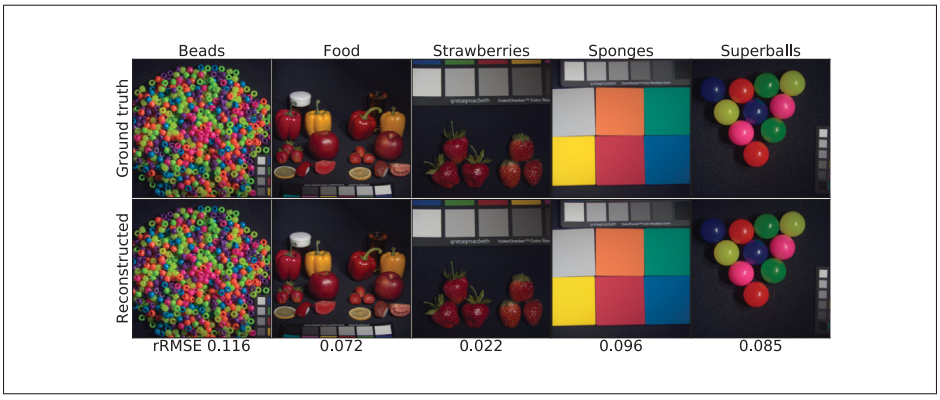

Figure 5.9 The effects of reconstructing the 5 chosen RGB images from the multispectral images

the error images. The blue represents the positive error, the green signifies the zero error, and the red denotes the negative error. Furthermore, we noticed that our MSI reconstruction images are clearer than those on the five selected bands compared to the same qualitative experiment conducted by Kin et al.

Moreover, with Generator G_{RGB}, we can quickly reconstruct RGBs from MSIs. Five of the same images, "Beads", "Food", "Strawberries", "Sponges", and "Superballs" were selected to evaluate the reconstruction quality of G_{RGB} in Figure 5.9. Since we cannot evaluate our result and that of Kin et al. by naked eye, we had to calculate every scene's relative RMSE and list them under the figure's bottom line. The result is better than that of Kin et al. (Gwn Lore *et al.*, 2019).

5.5.2.3 The quantitative measurement

After a full investigation on the measurements of related research, we decided to adopt the Structural Similarity Index (SSIM), Peak Signal-to-Noise Ratio (PSNR), and Root Mean Square Error (RMSE) as the main quantitative metrics.

The dataset processing has been explained previously. We rotated odd position samples and even position samples to train and test the model. Additionally, the training and testing were performed three times and the average value was calculated. We conducted the experiments in two directions, RGB to MSI and MSI to RGB and compared our results with the following three state-of-the-art works: Berk et al. (Kaya *et al.*, 2018), Kin et al. (Gwn Lore *et al.*, 2019), and Arad et al. (Arad & Ben-Shahar, 2016). The final results are listed in Tables 5.6 and 5.7, respectively.

Table 5.6 The quantitative measurement of converting RGB image to multispectral image

	Berk	**Kin**	**Arad**	**Ours**
Method	CNNs	cGANs	Sparse coding	**TaijiGNN**
Percentage of training and testing	-	50%:50%	-	**50%:50%**
RMSE~(0-255)	-	8.0622	**5.4**	5.656
RMSE~(0-1)	0.038	-	-	**0.022**
SSIM	0.94	-	-	**0.975**
PSNR	28.78	-	-	**33.91**

Table 5.7 The quantitative measurement of converting multispectral image to RGB image

	Berk	**Kin**	**Ours**
Method	CNNs	cGANs	**TaijiGNN**
Percentage of training and testing	-	50%:50%	**50%:50%**
RMSE~(0-255)	2.55	5.649	**1.727**
RMSE~(0-1)	0.038	-	**0.0068**
SSIM	0.94	-	**0.99**
PSNR	28.78	-	**45.07**

From Table 5.6, a phenomenon is noticeable, the results of Arad et al. ranked first and our results ranked second are very close. Our explanations are as follows:

First, Arad et al. did not clearly describe how they split the CAVE dataset for training and evaluation. According to our speculation, their splitting ratio should be 70–80% for training and 20–30% for testing. As shown in Figure 5.7, model training with more data usually yields a higher performance. Therefore, their splitting ratio gives their model more advantages.

Second, before training, the method of Arad et al. must first take a hyperspectral prior sampled from each image. The CAVE dataset's sampling ratio is 3.8% for every image, which is equivalent to knowing the answers to some questions before testing. This kind of method would be helpful to highly improve their model's effectiveness. However, it would easily lead to overfitting, causing it to be useless in reconstructing images in the unknown dataset.

On the contrary, Kin et al. adopted the same splitting dataset ratio as us, which was half for testing and half for training, and isolated the testing data and the training data strictly, which indicated that the model during training would not receive any information from the testing data. Therefore, TaijiGNN would have better performance in the unseen dataset. Therefore, it is not accurate to conclude that the approach of Arad et al. is superior to ours.

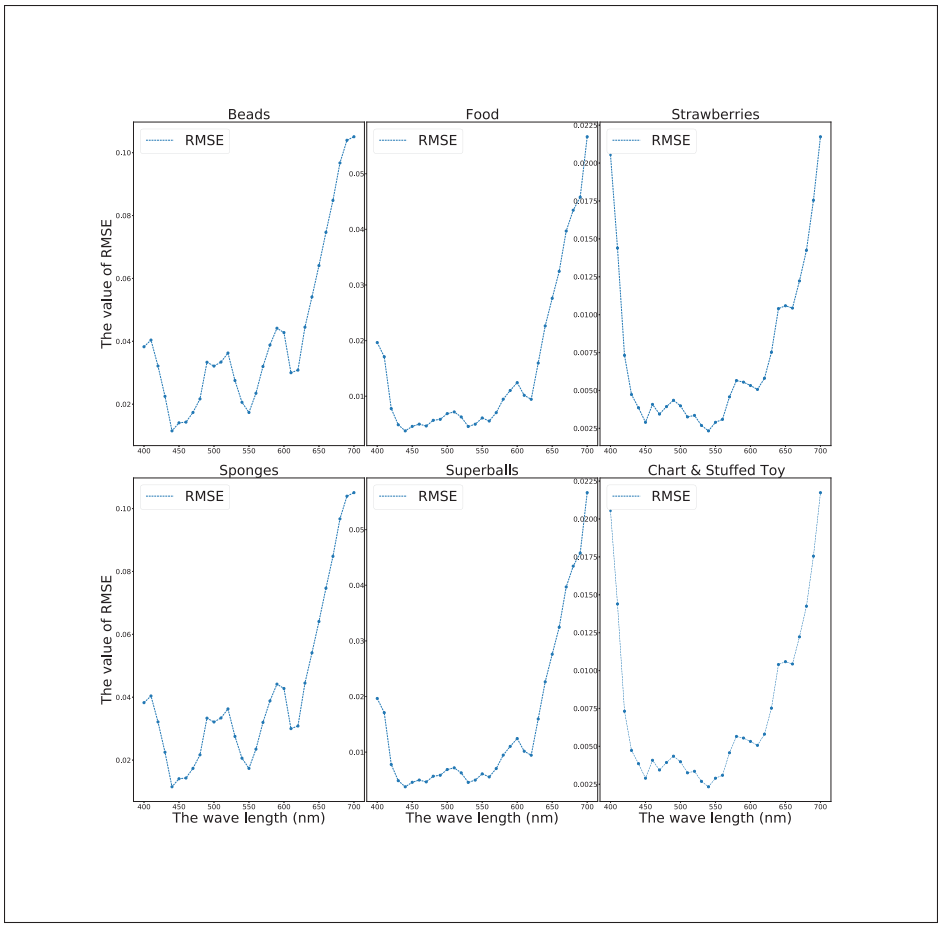

Figure 5.10 The RMSE wavelines of multispectral image restoration with the CAVE database

Moreover, Kin et al. adopted the same dataset splitting ratio as us. Compared with their RMSE result, TaijiGNN was reduced 30% in converting the RGB image to the multispectral image and reduced 69% in the reverse direction.

Furthermore, identical to Kin et al., we selected six different scenes, "Food", "Beads", "Sponges", "Strawberries", "chart & Stuffed Toy", and "Superballs", to investigate the reconstruction performance changing with the scene and the wavelength. Figure 5.10 demonstrates the RMSE changes in the six scenes.

From Figure 5.10, we can find that the model reconstruction performances have some differences in various scenes. However, the shapes of the RMSE curves are similar. Generally, the

generator performs better when reconstructing the band images whose wavelengths are 450 nm–650 nm. We provide the following reasons for this: From the CIE 1931 color matching function (Figure 5.2), we may notice that most red, green, and blue components are distributed mainly in the wavelength range of 450 nm–650 nm and allocate scarce elements to the two ends of the spectrum. One pixel of RGB image comprises three primitive colors, red, green, and blue, containing limited information about the two ends of the spectrum that may cause bad reconstructing behavior near the two-end spectral zone.

5.5.3 The ICVL Dataset

5.5.3.1 The Dataset Processing and the Hyperparameters Setting

The ICVL dataset was created by the interdisciplinary Computational Vision Lab of Ben-Gurion University. Its latest version consists of 201 scenes, most of which were taken outdoors by a Kappa DX4 hyperspectral camera. Additionally, the dimensions of every multispectral image are 1392 pixel height, 1300 pixel width, and 519 band depth. The 519 bands are distributed from 400 nm to 1000 nm in approximately 1.25 nm increments (Arad & Ben-Shahar, 2016).

In this experiment, we adopted a reduced version of the ICVL dataset, supplied by the authors (Arad & Ben-Shahar, 2016). The 519 bands were reduced to the 31 bands with approximately 10 nm intervals from 400 nm to 700 nm, which has two benefits: decreasing the computation cost and comparing them to former baselines easily (Arad & Ben-Shahar, 2016). Moreover, we used the CIE 1964 color matching function to generate the RGB images from the 519 band MSIs and the 31 band MSIs. The difference between them is too small to identify. The RMSE of the two kinds of generated RGB images is approximately 0.00014.

First, we randomized the order of the 201 scenes and divided them into six groups. Each of the first five groups, 1st, 2nd, 3rd, 4th, and 5th, had 32 samples. However, the 6th group contains 41 samples. We iterated one of the 1st–5th groups to train the model and the remaining four groups plus the 6th group to test the model. For instance, when taking the 3rd group to train the

model, the 1st, 2nd, 4th, 5th, and 6th groups were used to test the model. This kind of dataset processing aims to make the evaluation thorough and non-overlapping. Meanwhile, the splitting ratio of training and testing for the dataset reached 32:169, which means thirty-two samples to train the model and one-hundred-and-sixty-nine samples to test the model.

Additionally, as in Section 5.5.2.1, every 32 image training group was further separated into two equal-size parts. Images with even indexes, $0, 2, 4, \ldots, 30$ were placed into one group, whereas images with odd indexes, $1, 3, 5, \ldots, 31$ were placed into another group. We rotated the odd group and the even group to train and test the model.

Additionally, a further challenge was made to decrease the spatial resolution of training samples from [1392 (height), 1300 (width)] to [512 (height), 512 (width)]. To achieve the above target, we separate the original sample ([1392,1390]) into four equally sized ([512, 512]) blocks, top-right, top-left, bottom-right, and bottom-left. For each time training, we randomly selected one of the four blocks. In our experiment, we performed the training four times to ensure all four blocks would be chosen. The final training group's dimensions became [32 (sample), 512 (height), 512 (width), 34 (MSI+RGB bands)]. However, to compare the state-of-the-art works equally, the testing group's dimensions were still $[169, 1392, 1300, 34]$. Therefore, the ratio of the training samples and the testing samples achieved 32:169. As far as we know, until now, no one adopted such a small splitting ratio of training and testing in the ICVL dataset.

Furthermore, to completely assess the approach, bi-directional conversions between the multi-spectral image and the RGB image were performed. Additionally, qualitative and quantitative assessments were conducted simultaneously. Additionally, the works of Zhiwei et al. (Xiong *et al.*, 2017), Arad et al.'s (Arad & Ben-Shahar, 2016), and Berk et al.'s (Kaya *et al.*, 2018) were selected as references. Moreover, Zhiwei et al. declared that their HSCNN's outcomes surpassed those of other works (Xiong *et al.*, 2017).

Similarly, the detailed training parameter settings were as follows: the training epochs were 300, and the training and inferencing batch sizes were both 10,240. We adopted the Adam optimizer, whose learning rate is 2×10^{-4}.

5.5.3.2 The qualitative evaluation

The ICVL dataset reconstruction qualitative evaluation consists of two stages: reconstructing the MSIs from the RGBs and the reverse. Before the experiment, the training data and the testing data were effectively processed according to Section 4.5.3.1.

As the ICVL database was created by Arad et al., who claimed that their approach obtained superior results (Arad & Ben-Shahar, 2016). Therefore, the work of Arad et al. was chosen to be the main baseline. Figure 5.11 was created with Generator F_{MSI} according to the format of Figure 4 in the work of Arad et al (Arad & Ben-Shahar, 2016). Scene "BGU_0403-1419-1" and scene "prk_0328-1025" were selected as the samples. Moreover, 620 nm, 540 nm, and 460 nm bands were selected to exhibit the qualitative restoration performance. The top-left 1st RGB-image "prk_0328-1025" and 2nd RGB-image "BGU_0403-1419-1" in Figure 5.12 are the two original input RGBs.

The error maps were generated by subtracting the original image from the forecast image. Additionally, we set the colormap "jet" mode to pseudocolor the error images. The blue represents the positive error, the green signifies the zero error, and the red denotes the negative error.

Furthermore, when we compare the two ICVL figures [5.11,5.12] with the two previous CAVE figures [5.8,5.9], respectively, we may notice that the identical method performs more effectively on the ICVL scenes than on the CAVE scenes. We think that the main reason for this is that since most of the CAVE images are taken indoors, only a few bright objects contain useful information, and the main dark background contains scarce data to train the model. On the contrary, most of the ICVL images were captured outdoors. In addition to the objects, the background also includes a large amount of useful information that would help to improve the model training performance.

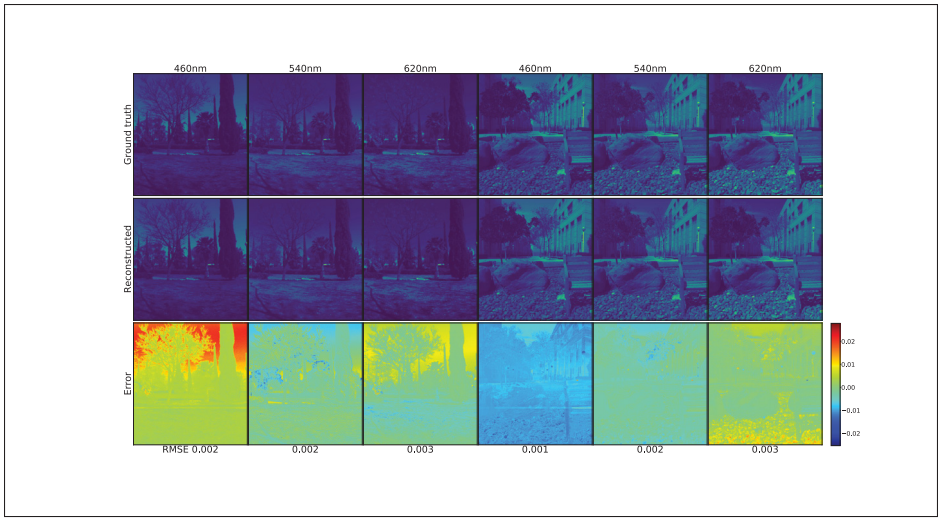

Figure 5.11 The effects of reconstructing the 3 chosen ICVL spectrum bands from the 2 scenes (The two original input RGB images are shown in the top-left 2 images of Figure 5.12.)

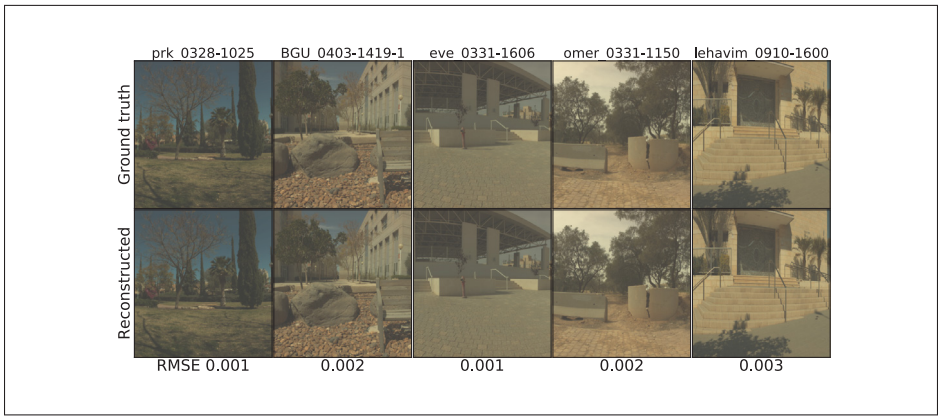

Figure 5.12 The effects of reconstructing the 5 chosen ICVL RGB images from the multispectral images

5.5.3.3 The quantitative measurement

Similar to the evaluation of the CAVE model, we picked four standard quantitative measurements, RMSE, PSNR, SSIM, and added relative RMSE (nRMSE or rRMSE) to evaluate the ICVL model's performance.

Moreover, we prepared two experiments and several comparisons with the leading related works to show TaijiGNN's advantages.

We compared the results of TaijiGNN with those of the two top works, the HSCNN by Zhiwei et al. and the sparse coding by Arad et al., during the first experiment. To make the comparison impartial, we adopted an equal or higher level to process the ICVL dataset, which is described in the following paragraphs.

Arad et al. performed all their experiments on a dataset named "complete set", owning 102 scenes. After careful comparison, we noticed that in their supplied dataset, scene "lst_0408-0924" lost (Arad & Ben-Shahar, 2016). Therefore, we removed it from the dataset. The final scene amount of the dataset became one-hundred-and-one. Additionally, they selected 59 scenes from the dataset and assigned them to 5 domains: Indoor, Park, Rural, Urban, and Plant-life. Their experiment strategy is domain-specific, selecting a scene from a domain for testing, and the remaining scenes of the domain were used to train the model. Additionally, the above steps were repeated until all scenes of that domain had been picked. Finally, Arad et al. measured the model's performance by the average relative RMSE (rRMSE). According to their claim, the model trained by the domain-specific subsets has better results than the complete dataset model (Arad & Ben-Shahar, 2016).

Zhiwei et al. processed the ICVL dataset more generally when evaluating their approach, "HSCNN". Their dataset contains 200 images, including the complete set of previous images. Their experiment has two phases: In the first phase, they trained the model with 141 images and tested on the 59 domain-specific images. In the second phase, they trained the model with the 159 images, excluding the non-domain-specific images in the complete set, and tested the obtained model on the remaining 41 images. In this way, the training images and the testing images were precisely isolated, and the domain-specific restriction was eliminated (Xiong *et al.*, 2017).

We extended the dataset splitting method of Zhiwe et al. First, we updated the total dataset to 201 images. Second, we reduced the training image number to 100 images and assessed the neural

network on the complete set (101 scenes) of Arad et al., containing the fifty-nine domain-specific scenes. The training and testing images were rigorously isolated. Third, in each evaluation, we randomly sampled 32 images from the 100 training images and decreased their spatial size from [1392 (height), 1390 (width)] to [512 (height), 512 (width)]. The detailed process is as follows: First, we separate the original image ([1392, 1390]) into four equally sized ([512, 512]) frames, top-right, top-left, bottom-right, and bottom-left, to achieve the above-mentioned spatial size translation. For each time training, we randomly selected one of the four blocks. Although the training image spatial resolution was 512×512, the test images maintained the 1392×1300 spatial resolution, which is one of our approach's advantages. To thoroughly cover all the images, we performed the evaluation 5 times and calculated their average values.

Table 5.8 summarizes the above three approaches' results of reconstructing MSIs from RGBs. Our approach, TaijiGNN, surpasses the other two in most subsets and the complete set. Furthermore, our training and testing ratio is the lowest.

Table 5.8 The three approach results of reconstructing MSIs from RGBs on the ICVL dataset

Author	Arad et al. (Arad & Ben-Shahar, 2016)	Zhiwe et al. (Xiong *et al.*, 2017)	Ours
Methods	Sparse coding	HSCNN	TaijiGNN
Percentage of training and testing	-	141:59	**100:101**
Evaluation metric	rRMSE	rRMSE	rRMSE
Park subset	0.0589	0.0371	**0.0347**
Urban subset	0.0617	0.0388	**0.0335**
Indoor subset	0.0507	0.0638	**0.0495**
Plant-life subset	0.0469	0.0445	**0.0434**
Rural subset	0.0354	**0.0331**	0.0369
Subset average	0.0507	0.0435	**0.0396**
Complete set	0.0756	0.0388	**0.0358**

We increased the challenge in the second experiment to fully explore TaijiGNN's potential. We removed the first experiment dataset splitting method and adopted the dataset splitting method in Section 4.5.3.1.

We randomized the order of the 201 scene samples and divided them into six groups. Each of the first five groups, 1st, 2nd, 3rd, 4th, and 5th, had thirty-two samples. However, the 6th group

contained forty-one samples. We iterated one of the 1st–5th groups to train the model and the remaining four groups with the 6th group to test the model. For instance, when taking the 3rd group to train the model, the 1st, 2nd, 4th, 5th and the 6th groups were used to test the model. This dataset processing aims to make the evaluation thorough and non-overlapping. Meanwhile, the splitting ratio of training and testing for the dataset reached 32:169, which means thirty-two samples to train the model and one-hundred-and-sixty-nine samples to test the model.

Additionally, every 32-image training group was further separated into two equally sized parts. Images with even indexes, $0, 2, 4, \ldots, 30$ were placed into one group, whereas images with odd indexes, $1, 3, 5, \ldots, 31$ were placed into another group. We rotated the odd group and the even group to train and test the model.

Additionally, we decreased the spatial resolution of the training samples from [1392 (height), 1300 (width)] to [512 (height), 512 (width)] to increase the training speed, which unavoidably increased the convergence difficulty. A detailed description of the dataset processing is presented in the previous paragraphs. The final training group dimension bacame $[32, 512, 512, 34]$. However, the testing group dimension maintained $[169, 1392, 1300, 34]$. Therefore, the splitting ratio of the training group and the testing group achieved 32:169. To make the evaluation cover the whole dataset, we conducted four experiments on every group and calculated their average values.

We present the final bi-directional conversion results between MSIs and RGBs in Table 5.9. To the best of our knowledge, such a training and testing experiment with such a low splitting ratio has not been conducted to date. Compared with the CAVE dataset outcomes (Tables 5.6 and 5.7), we can immediately discover that the performance of TaijiGNN is much higher in the ICVL database than in the CAVE database, which affirms the previous analysis in Section 5.5.3.2.

Moreover, to investigate the rule of RMSE changing with the scenes and the light wavelength, image "prk_0328-1025", image "BGU_0403-1419-1", and image "eve_0331-1606" were chosen to assess the 31 band multispectral restoration RMSEs, respectively, and the corresponding RMSE curves were drawn in Figure 5.13. Additionally, Arad et al. adopted the image "prk_-

Table 5.9 The bi-directional translation outcomes of TaijiGNN on the ICVL database

TaijiGNN Train(32 samples):Test(169 samples)					
Metrics	**SSIM**	**PSNR**	**rRMSE**	**RMSE**	**RMSE_INT**
MSI to RGB	1.000	54.994	0.004	0.002	0.492
RGB to MSI	0.999	44.665	0.029	0.007	1.700

0328-1025" and image "BGU_0403-1419-1" in their paper (Arad & Ben-Shahar, 2016) and image "eve_0331-1606" was adopted in the work of Zhiwei et al. as an example (Xiong *et al.*, 2017).

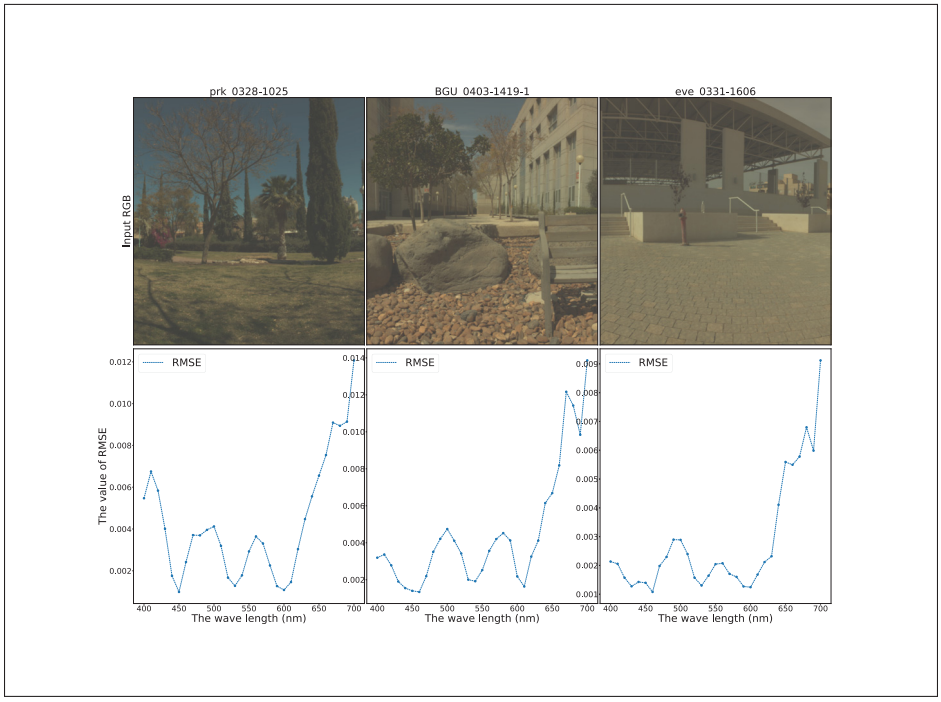

Figure 5.13 The RMSE waveline of multispectral image restoration on the ICVL database

Observing Figure 5.13, we can notice that the model's reconstruction performance performs poorly within the long wavelength spectrum. Our explanation is as follows: Since the three images were taken outdoor in daylight, the backgrounds and environments may generate large amounts of thermal noise and long-wave light, disturbing the collection of useful long-wave light

from objects. When we recorded the long-wavelength light in the daytime, we simultaneously saved a large amount of background noise, mainly from the sunlight and the background. Too much background noise cuts down the MSI reconstructing performance.

5.6 Limitations

TaijiGNN has four losses, $\mathcal{L}_{RGB}$ (5.5), $\mathcal{L}_{MSI}$ (5.6), $\mathcal{L}_{cycle\_MSI}$ (5.7), and $\mathcal{L}_{cycle\_RGB}$ (5.8). We took advantages of these losses to train Generator G_{RGB} and Generator F_{MSI}. In the experiment section, we only set a 1:1:1:1 ratio for the four losses. However, we are not confident that the 1:1:1:1 ratio is the best, as we did not fully understand their roles during the training process.

To make clear the roles of the four losses, we maintained the values of $\mathcal{L}_{cycle\_MSI}$ (5.7) and $\mathcal{L}_{cycle\_RGB}$ (5.8), and multiplied parameter β by $\mathcal{L}_{MSI}$ (5.6) and $\mathcal{L}_{RGB}$ (5.5). In this way, we obtained the total loss formula (5.9). Moreover, we tuned the β value to observe the RMSE changing against the β.

$$\mathcal{L}_{Totally} = \beta \times (\mathcal{L}_{MSI} + \mathcal{L}_{RGB}) + \mathcal{L}_{cycle\_MSI} + \mathcal{L}_{cycle\_RGB} \tag{5.9}$$

We performed two experiments on the CAVE dataset. One is reconstructing MSIs from RGBs, and the other is reconstructing RGBs from MSIs. The results are listed in Tables 5.10 and 5.11, respectively.

Table 5.10 The RGB to MSI RMSE changes against the β variations

β	**PSNR**	**SSIM**	**RMSE (0-1)**	**RMSE (0-255)**
0.01	20.149	0.6944	0.1073	27.368
0.1	33.383	0.9692	0.0247	6.302
0.5	33.821	0.9731	0.0232	5.915
1	33.683	0.9727	0.0234	6.010
2	33.782	0.9728	0.0234	5.957
3	33,767	0.9728	0.0232	5.966
CAVE dataset, train_epoch=300, batch_size=10240				

Table 5.11 The MSI to RGB RMSE changes against the β variations

β	**PSNR**	**SSIM**	**RMSE (0-1)**	**RMSE (0-255)**
0.01	16.672	0.7793	0.1569	40.013
0.1	38.261	0.9784	0.0136	3.467
0.5	46.318	0.9919	0.0054	1.381
1	46.273	0.9911	0.0056	1.423
2	50.557	0.9924	0.0034	0.875
3	50.955	0.9953	0.0032	0.809
CAVE dataset, train_epoch=300, batch_size=10240				

Moreover, the two tables' contents are shown in lines in Figure 4.12. From Figure 4.12, we can quickly conclude that loss $\mathcal{L}_{MSI}$ (5.6) and loss $\mathcal{L}_{RGB}$ (5.5) play larger roles in the training process. When we decreased their percentages, the performances of the two generators decreased quickly.

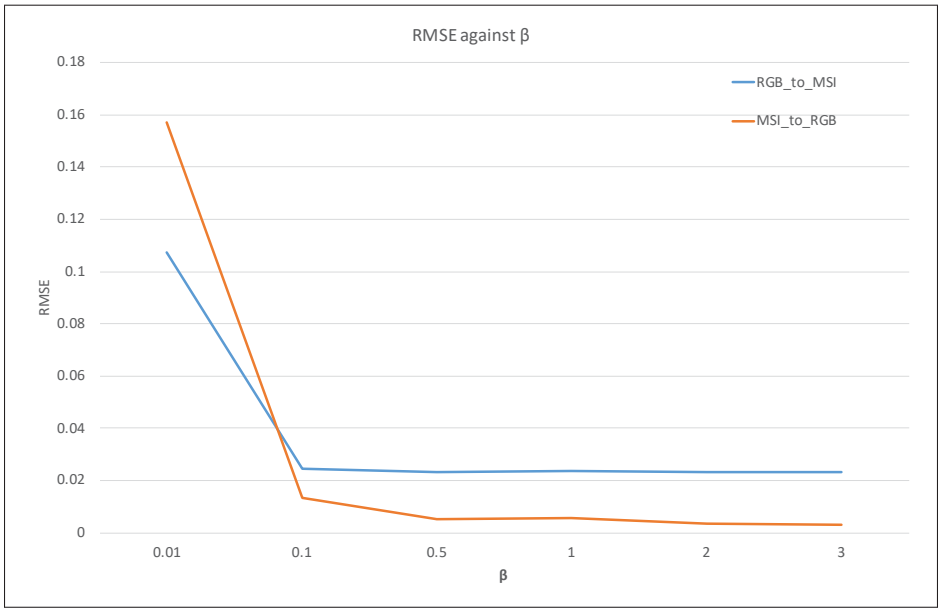

Figure 5.14 The RMSE against the β on the CAVE dataset

Moreover, since this is our first time investigating the feasibility of using the cycled neural network to translate between MSIs and RGBs, we adopted a symmetrical architecture for simplicity. Generator G_{RGB} and Generator F_{MSI} adopt the same neural network structures. However,

we know that reconstructing MSIs from RGBs is much more complex than reconstructing RGBs from MSIs. It is an asymmetrical problem. The symmetrical structure may restrict the performance of the neural network. We will investigate asymmetrical neural network architecture to improve the performance of TaijiGNN in the future.

Furthermore, if the spectral range is above 700 nm, it can be reconstructed by RGB. However, it needs corresponding multispectral images, which are taken above 700 nm. Currently, both the CAVE dataset and the ICVL dataset are taken within 400 nm–700 nm. Since both the datasets do not have any information above 700 nm, it is impossible to reconstruct the multispectral images above 700 nm from the RGB of the CAVE dataset and the ICVL dataset. The CAVE and the ICVL datasets are the most popular datasets to verify spectral super-resolution performance. Since it is not easy to find a suitable dataset above 700 nm, but few related works have done it, it is hard to find a baseline to compare and verify our approach. We plan to extend this study and evaluate our work above 700 nm and below 400 nm in the future.

5.7 Conclusions and future work

In this paper, we first introduced the two challenges in the bi-directional conversion between MSIs and RGBs. One challenge is that MSIs and RGBs belong to two different domains and have different rigid definitions, and direct translation is difficult to accomplish. The other is that reconstructing MSIs from RGBs is a severely underconstrained process due to the colossal information entropy gap.

To address the two above-mentioned challenges, we proposed a new approach, "TaijiGNN", which can convert the problem of comparing different domain images into comparing the same domain images by the cycle neural network architecture. In this way, the two above-mentioned problems can be solved naturally. In addition, we used a well-designed multilayer perceptron neural network to substitute the convolutional neural network when implementing the generators to make them simpler and more efficient. Besides, we cut off the two traditional CycleGAN

identity losses to fit the spectral image translation and added two consistency losses to enhance the model training.

Finally, several experiments were conducted on the two classical datasets, CAVE and ICVL, to evaluate our method thoroughly. Under the same validation configuration as the previous state-of-the-art, much less training data enable our approach to obtain similar outcomes in the CAVE dataset and gain dramatic improvements in the ICVL dataset.

In the future, we plan to extend our work in the following directions: First, we will investigate asymmetrical neural network architecture to improve the performance of TaijiGNN and the impacts of the hyperparameter change. Second, we will extend our approach to the spectral image dataset above 700 nm and below 400 nm. Third, we will use GPU and FPGA to accelerate the training and inferencing processes and investigate the performance improvements as we did in the previous works (Liu *et al.*, 2019b) and (Liu *et al.*, 2018a). Last but not least, since the spectrum is a kind of sequence data, we will explore using Long short-term memory (LSTM) to improve the performance of reconstructing the multispectral images (Liu *et al.*, 2019a).

5.8 Appendices

Table 5.12 shows the performance summary of TaijiGNN on the CPU and GPU. From the table, we can find that the GPU speed is about 10× faster than the CPU, which is consistent with our previous works, (Liu *et al.*, 2019b) and (Liu *et al.*, 2018a).

Table 5.12 The performance summary of TaijiGNN on the CPU and the GPU

Batchsize = 10240		**CAVE(CPU)**	**CAVE(GPU)**	**ICVL(CPU)**	**ICVL(GPU)**
Training image size		RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)
Average one epoch training time (s)		113	8	7	113
Inferencing image size		RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(1, 1392, 1300, 3) MSI(1, 1392, 1300,31)	RGB(1, 1392, 1300, 3) MSI(1, 1392, 1300,31)
Average image inferencing time (s)	RGB2MSI	9.5	1.1	4.1	0.4
	MSI2RGB	9.2	1.1	4.1	0.4

CHAPTER 6

GENERAL DISCUSSION

Since this thesis is based on three published journals, this chapter focuses on introducing their links.

The three articles involve two different fields: high-performance computing and image processing. Specifically, "A hybrid GPU-FPGA based design methodology for enhancing machine learning applications performance" (Hybrid ML) proposed a novel heterogeneous platform to improve machine learning performance and applied the platform to handwritten digit recognition and power usage effectiveness prediction. "Multispectral Image Reconstruction From Color Images Using Enhanced Variational Autoencoder and Generative Adversarial Network" (VAE-GAN) and "TaijiGNN: A New Cycle-Consistent Generative Neural Network for High-Quality Bidirectional Transformation between RGB and Multispectral Domains" (TaijiGNN) focused on solving severely under-constrained spectrum reconstruction problems.

On the surface, the three articles seem to have few relations. However, in fact, they are closely connected. Figure 6.1 shows the link among the three articles.

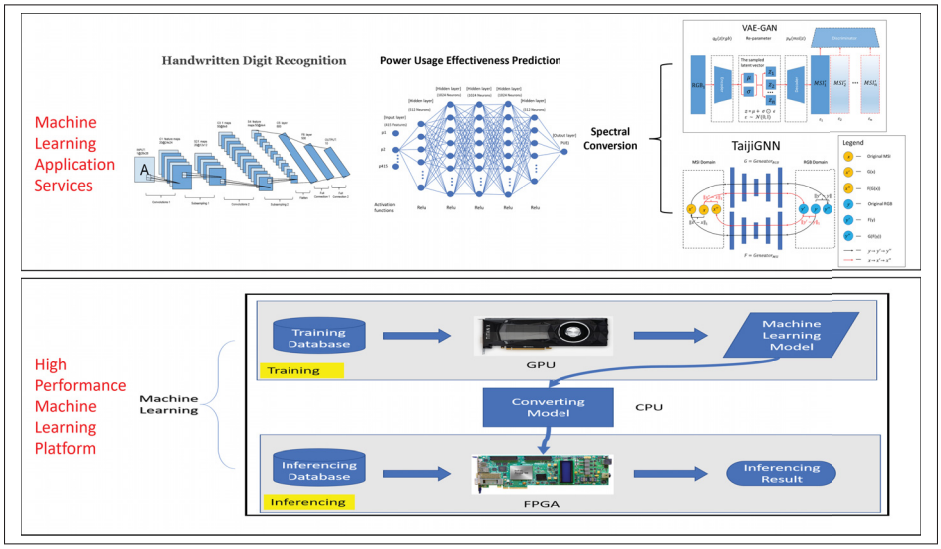

Figure 6.1 The High Performance Machine Learning Stack

Figure 6.1 shows the connection among the three articles. As described in Chapter 2, the main objective is to build a heterogeneous platform that can improve ML application performance and apply the platform to ML applications. Refer to the current mainstream naming rules, and I named it "High Performance Machine Learning Stack (HPMLS)".

The HPMLS consists of two parts, high performance machine learning platform and machine learning application services. Since the paper "Hybrid ML" has elaborated on how to design and develop a high performance machine learning platform and apply the platform to handwritten digit recognition and PUE prediction, I do not repeat it here. I will give a more explanation about what is the role of the platform in the paper "VAE-GAN" and the article "TaijiGNN".

Both of the two papers' main targets are to do bi-directional conversion between the RGB domain and the ML domain. Sixteen training samples with 512 x 512 x 34 resolution will be thrown into a neural network with 3x512x512x31 neurons to calculate during each training epoch, which requires high-density computing power. As analyzed in the paper "Hybrid ML", traditional computing devices like CPUs cannot meet this high-density computing requirement. We need to use other hardware devices like GPUs or FPGAs to accelerate the training and inferencing process.

Figure 6.2 shows the detailed architecture of the spectral conversion stack. The biggest difference between Figure 6.2 and Figure 6.1 is Figure 6.2 used GPU to replace the FPGA of Figure 6.1 during the inference stage. That was because currently, I focused on improving the spectral conversion algorithms and often tried different algorithms. It is much easier to implement an algorithm of the same scale using GPU than using FPGA. So I chose GPU to accelerate both the training and inferencing processes in the spectral conversion. In addition, because the training and inference processes use the same hardware architecture, no model conversion is required, so the model conversion module is eliminated.

The paper "VAE-GAN" and "TaijiGNN" only have differences in the algorithms and effectiveness. However, they have the same problems to be solved, the same experimental environment, and

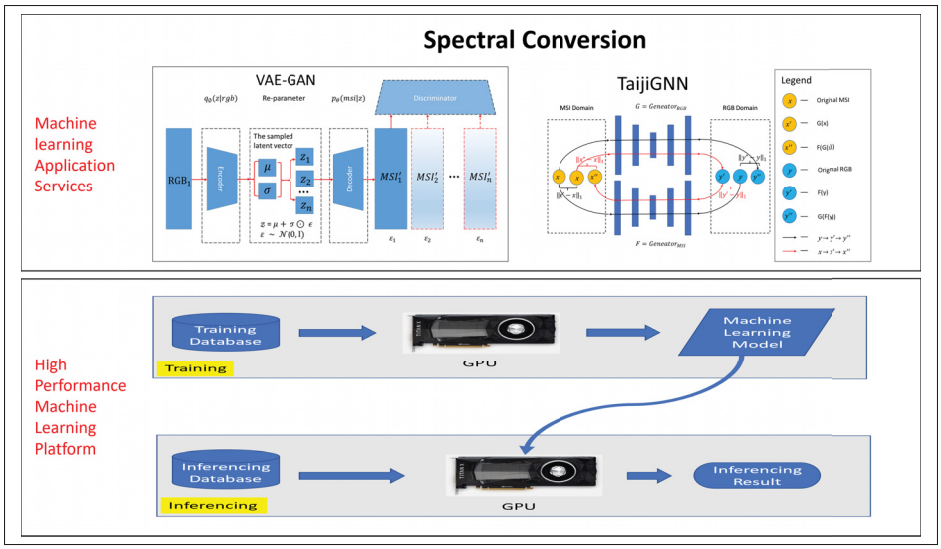

Figure 6.2 The Spectral Conversion Stack

the same experimental evaluation methods, so we only take the paper "TaijiGNN" as an example to introduce in detail.

In the paper "TaijiGNN", I have measured the CPU's and GPU's training and inferencing time on CAVE and ICVL datasets, respectively. Table 6.1 shows the performance summary of TaijiGNN on the CPU and GPU. From the table, we can find that the GPU speed is about 10× faster than the CPU, which is consistent with our previous works, (Liu *et al.*, 2019b) and (Liu *et al.*, 2018a). In addition, GPU has also achieved similar acceleration effects in the training and inferencing part of the paper "VAE-GAN".

In the future, when the development of the spectral conversion algorithm is completed, it is possible to study how to use FPGA to further accelerate the inferencing speed. The ultimate goal is to create a stack of various machine learning application services based on heterogeneous computing hardware in Figure 6.1.

Table 6.1 The performance summary of TaijiGNN on the CPU and the GPU

Batchsize = 10240		**CAVE(CPU)**	**CAVE(GPU)**	**ICVL(CPU)**	**ICVL(GPU)**
Training image size		RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)
Average one epoch training time (s)		113	8	7	113
Inferencing image size		RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(16, 512, 512, 3) MSI(16,512,512,31)	RGB(1, 1392, 1300, 3) MSI(1, 1392, 1300,31)	RGB(1, 1392, 1300, 3) MSI(1, 1392, 1300,31)
Average image inferencing time (s)	RGB2MSI	9.5	1.1	4.1	0.4
	MSI2RGB	9.2	1.1	4.1	0.4

CONCLUSION AND RECOMMENDATIONS

With machine learning booming, more and more challenges appear. This thesis starts by introducing the machine learning algorithms, the performance challenges, and the application problems. And then it gave its methodologies.

The thesis introduces some machine learning foundational concepts, history, and main ML algorithms in the literature review chapter. It also reviews some related work about high performance machine learning and introduces some fundamental knowledge in the spectral conversion field.

And the first article analyzed the causes of the challenge and concluded that the traditional CPU architecture is not fit for machine learning algorithms. Next, the thesis introduced two new hardware, GPU and FPGA, which could boost machine learning performance. And it proposed a novel heterogeneous machine learning platform based on GPU and FPGA. This platform consists of a GPU training part, an FPGA inferencing part, and a model converting part. Moreover, the article implemented a CNN and a DNN to verify the performance of the heterogeneous machine learning platform. The results demonstrated the heterogeneous platform performance is superior to uni-CPU, uni-GPU, or uni-FPGA implementations.

In the second article, a multispectrum reconstruction approach based on VAE-GAN was presented. This approach combines VAE and GAN in one neural network. The VAE can abstract the input's critical features and merge the features with random variables to create new latent vectors that could be used as the following GAN input. And the GAN can use its discriminator to help the generator to create the desired distribution. In this way, one input could yield multiple variants mapping to the multiple outputs. The underconstrained spectrum reconstruction problems could be solved quickly. And the experiment results also proved this approach reaches the state of arts.

The third article proposed a new neural network, "TaijiGNN," to solve the spectrum conversion problems. The ancient Chinese philosophy's "Taiji" concept and cycled generative neural network architecture were merged into one neural network. The cycled architecture can transfer the input via the output domain back to the input domain. In this way, TaijiGNN successfully turns the problem of comparing different domain images into comparing the same domain images. Besides, the two generators of TaijiGNN comprise and complement each other during the training process, which is very similar to Taiji's two poles, "Yang" and "Yin" working mechanism. Moreover, TaijiGNN behaved state of the art performance in the experiments.

Although the above articles got some preliminary positive results, numerous improvements and new fields could be investigated in the future. Some of the future works are presented in the following paragraphs.

Currently, we need to use different programming languages to develop applications for the GPU and the FPGA. It is not convenient and efficient. So unifying the programming languages is a meaningful job. Besides, since the CPU, the GPU, and the FPGA are separated on dedicated boards, they frequently need to share data among the accelerating devices through the peripheral component interconnect express (PCIe), which becomes the bottleneck of performance improvement. A good solution may be integrating all devices on one board and unifying the memory. In this way, all devices can communicate information by the shared memory, saving considerable bandwidth and significantly boosting system performance.

Addressing the spectrum reconstruction, currently, VAE-GAN and TaijiGNN worked well on the CAVE and ICVL data sets. However, the two datasets are both well cleaned and processed by the suppliers and are very suitable for spectrum experiments. In the actual scenarios, it is impossible to get such perfect datasets. Most real images would have different spatial resolutions and noise. So it is meaningful to evaluate the two approaches on the images taken by the consumer cameras.

Furthermore, I hope these two approaches can be developed into applications and transplanted on our daily used devices like mobile phones and help us improve our life quality.

APPENDIX I

LIST OF PUBLICATIONS

1. Journals

1.1 Published

1. X. Liu, HA. Ounifi, A. Gherbi, W. Li and M. Cheriet, "A hybrid GPU-FPGA based design methodology for enhancing machine learning applications performance", in Journal of Ambient Intelligence and Humanized Computing (JAIHC), vol. 11, pp. 2309–2323, 2020, doi:10.1007/s12652-019-01357-4.

2. X. Liu, A. Gherbi, Z. Wei, W. Li and M. Cheriet, "Multispectral Image Reconstruction From Color Images Using Enhanced Variational Autoencoder and Generative Adversarial Network", in IEEE Access, vol. 9, pp. 1666-1679, 2021, doi: 10.1109/ACCESS.2020.3047074.

3. X. Liu, A. Gherbi, W. Li, Z. Wei, and M. Cheriet, "TaijiGNN: A New Cycle-Consistent Generative Neural Network for High-Quality Bidirectional Transformation between RGB and Multispectral Domains", in MDPI Sensors, vol. 21, 2021, doi: 10.3390/s21165394.

2. Conferences

2.1 Published

1. Xu Liu, Hibat Allah Ounifi, Abdelouahed Gherbi, Yves Lemieux and Wubin Li, "A Hybrid GPU-FPGA-based Computing Platform for Machine Learning," The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018), Leuven, Belgium, 2018, vol. 141, pp. 104-111.
2. Xu Liu, Abdelouahed Gherbi, Wubin Li and Mohamed Cheriet, "Multi Features and Multi-time steps LSTM Based Methodology for Bike Sharing Availability Prediction,"

The 14th International Conference on Future Networks and Communications (FNC-2019), Halifax, Canada, 2019, vol. 155, pp. 394-401.

3. Hibat-Allah Ounifi, Xu Liu, Abdelouahed Gherbi, Yves Lemieux and Wubin Li, "Model-based Approach to Data Center Design and Power Usage Effectiveness Assessment," The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018), Leuven, Belgium, 2018, vol. 141, pp. 143-150.

3. Patents

1. Xu Liu, Wubin Li, Yves Lemieux, Abdelouahed Gherbi, Hibat-Allah Ounifi, "METHOD, APPARATUS AND SYSTEM FOR HIGH PERFORMANCE PERIPHERAL COMPONENT INTERCONNECT DEVICE RESOURCE SHARING IN CLOUD ENVIRONMENTS," Publication Number: WO/2020/245636, International Application No. PCT/IB2019/054737, International Filing Date 06.06.2019.

BIBLIOGRAPHY

Abraham, C. (2020). A Beginner's Guide to (CIE) Colorimetry. Retrieved from: https://medium.com/hipster-color-science/a-beginners-guide-to-colorimetry-401f1830b65a.

Advanced Micro Devices, Inc. and Xilinx, Inc. (2017, May). Unified Deep Learning with CPU, GPU, and FPGA Technologies. Retrieved from: WhitePaper.

Altosaar, J. (2020). Tutorial - What is a variational autoencoder? Retrieved from: https://jaan.io/what-is-variational-autoencoder-vae-tutorial/.

Amara, M. et al. (2018). Temperature and color management of silicon solar cells for building integrated photovoltaic. *EPJ Photovoltaics*, 9, 1. doi: 10.1051/epjpv/2017008.

Amazon Inc. (2021). Introducing Amazon Machine Learning. Retrieved from: https://aws.amazon.com/about-aws/whats-new/2015/04/introducing-amazon-machine-learning/.

Andersson-Engels, S., Johansson, J. & Svanberg, S. (1994). Medical diagnostic system based on simultaneous multispectral fluorescence imaging. *Appl. Opt.*, 33(34), 8022–8029. doi: 10.1364/AO.33.008022.

Arad, B. & Ben-Shahar, O. (2016). Sparse Recovery of Hyperspectral Signal from Natural RGB Images. *European Conference on Computer Vision*, pp. 19–34.

Arad, B. & Ben-Shahar, O. (2020). ICVL dataset. Retrieved from: "http://icvl.cs.bgu.ac.il/hyperspectral/".

Arad, B., Ben-Shahar, O., Timofte, R., Van Gool, L., Zhang, L. & Yang, M. H. (2018). NTIRE 2018 challenge on spectral reconstruction from RGB images. *31st Meeting of the IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops, CVPRW 2018*, pp. 1042–1051.

Arad, B., Timofte, R., Ben-Shahar, O., Lin, Y.-T., Finlayson, G., Givati, S. et al. (2020). NTIRE 2020 Challenge on Spectral Reconstruction from an RGB Image.

Aydonat, U., O'Connell, S., Capalija, D., Ling, A. C. & Chiu, G. R. (2017a). An OpenCL™ Deep Learning Accelerator on Arria 10. *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, (FPGA '17), 55–64. doi: 10.1145/3020078.3021738.

Aydonat, U., O'Connell, S., Capalija, D., Ling, A. C. & Chiu, G. R. (2017b). An OpenCL Deep Learning Accelerator on Arria 10. *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 55–64.

Baronti, S., Casini, A., Lotti, F. & Porcinai, S. (1998). Multispectral imaging system for the mapping of pigments in works of art by use of principal-component analysis. *Appl. Opt.*, 37(8), 1299–1309. doi: 10.1364/AO.37.001299.

Bauer, S., Köhler, S., Doll, K. & Brunsmann, U. (2010). FPGA-GPU Architecture for Kernel SVM Pedestrian Detection. *Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 61–68.

Bergstra, James et al. (2011). Theano: Deep Learning on GPUs with Python. *NIPS 2011, BigLearning Workshop, Granada, Spain*, 3, 0.

Bettoni, M., Urgese, G., Kobayashi, Y., Macii, E. & Acquaviva, A. (2017). A Convolutional Neural Network Fully Implemented on FPGA for Embedded Platforms. *New Generation of CAS (NGCAS)*, pp. 49–52.

Can, Y. B. & Timofte, R. (2018). An efficient CNN for spectral reconstruction from RGB images.

Chen, H. & Liu, Y. (2014). Chapter 2 - Teeth. In Shen, J. Z. & Kosmač, T. (Eds.), *Advanced Ceramics for Dentistry* (pp. 5-21). Oxford: Butterworth-Heinemann. doi: https://doi.org/10.1016/B978-0-12-394619-5.00002-X.

Chen, Cheng et al. (2019). MMdnn. Retrieved from: https://github.com/Microsoft/MMdnn.

Choi, I., Jeon, D. S., Nam, G., Gutierrez, D. & Kim, M. H. (2017). High-quality Hyperspectral Reconstruction Using a Spectral Prior. *ACM Trans. Graph.*, 36(6), 218:1–218:13. doi: 10.1145/3130800.3130810.

Choudhury, A. K. R. (2014). 4 - Principles of colour perception. In Choudhury, A. K. R. (Ed.), *Principles of Colour and Appearance Measurement* (pp. 144 - 184). Woodhead Publishing. doi: https://doi.org/10.1533/9780857099242.144.

Cover, T. & Hart, P. (1967). Nearest neighbor pattern classification. *IEEE Transactions on Information Theory*, 13(1), 21-27. doi: 10.1109/TIT.1967.1053964.

Da Silva, B., Braeken, A., D'Hollander, E. H., Touhafi, A., Cornelis, J. G. & Lemeire, J. (2013). Comparing and Combining GPU and FPGA Accelerators in an Image Processing Context. *2013 23rd International Conference on Field Programmable Logic and Applications (FPL).*, pp. 1–4.

Datacenterknowledge. (2019). Improving Electrical Efficiency in Your Data Center. Retrieved from: https://www.datacenterknowledge.com/archives/2014/09/23/improving-electrical-efficiency-data-center.

DeJong, G. & Lim, S. H. (2010). Explanation-Based Learning. In Sammut, C. & Webb, G. I. (Eds.), *Encyclopedia of Machine Learning* (pp. 388–392). Boston, MA: Springer US. doi: 10.1007/978-0-387-30164-8_296.

DiCecco, R., Lacey, G., Vasiljevic, J., Chow, P., Taylor, G. & Areibi, S. (2016). Caffeinated FPGAs: FPGA framework For Convolutional Neural Networks. *2016 International Conference on Field-Programmable Technology (FPT)*, pp. 265-268. doi: 10.1109/FPT.2016.7929549.

Dwight, J. G., Weng, C. Y., Coffee, R. E., Pawlowski, M. E. & Tkaczyk, T. S. (2016). Hyperspectral image mapping spectrometry for retinal oximetry measurements in four diseased eyes. *International ophthalmology clinics*, 56(4), 25.

Earnest, L. (2012). Stanford Cart. Retrieved from: https://web.stanford.edu/~learnest/sail/oldcart.html.

Edelman, G. & Aalders, M. (2018). Photogrammetry using visible, infrared, hyperspectral and thermal imaging of crime scenes. *Forensic Science International*, 292, 181 - 189. doi: https://doi.org/10.1016/j.forsciint.2018.09.025.

Edelman, G., Gaston, E., van Leeuwen, T., Cullen, P. & Aalders, M. (2012). Hyperspectral imaging for non-contact analysis of forensic traces. *Forensic Science International*, 223(1), 28 - 39. doi: https://doi.org/10.1016/j.forsciint.2012.09.012.

Eldredge, J. G. & Hutchings, B. L. (1994). RRANN: a Hardware Implementation of the Backpropagation Algorithm using Reconfigurable FPGAs. *Proceedings of the IEEE World Congress on Computational Intelligence*, 4, 2097–2102.

Ellrod, G. P. (1995). Advances in the Detection and Analysis of Fog at Night Using GOES Multispectral Infrared Imagery. *Weather and Forecasting*, 10(3), 606-619. doi: 10.1175/1520-0434(1995)010<0606:AITDAA>2.0.CO;2.

Ganesh, S. S., Arulmozhivarman, P. & Tatavarti, V. S. N. R. (2018). Prediction of PM2.5 using an ensemble of artificial neural networks and regression models. *Journal of Ambient Intelligence and Humanized Computing*, 0. doi: 10.1007/s12652-018-0801-8.

Gevaert, C. M., Suomalainen, J., Tang, J. & Kooistra, L. (2015). Generation of Spectral–Temporal Response Surfaces by Combining Multispectral Satellite and Hyperspectral UAV Imagery for Precision Agriculture Applications. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing*, 8(6), 3140-3146. doi: 10.1109/JSTARS.2015.2406339.

Gharakhanian, A. (2017). Generative Adversarial Networks. Retrieved from: https://www.kdnuggets.com/2017/01/generative-adversarial-networks-hot-topic-machine-learning.html.

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., Courville, A. & Bengio, Y. (2014). Generative Adversarial Nets. In Ghahramani, Z., Welling, M., Cortes, C., Lawrence, N. D. & Weinberger, K. Q. (Eds.), *Advances in Neural Information Processing Systems 27* (pp. 2672–2680). Curran Associates, Inc. Retrieved from: http://papers.nips.cc/paper/5423-generative-adversarial-nets.pdf.

Google Brain Team. (2018). The MNIST Matrix. Retrieved from: https://www.tensorflow.org/versions/r1.1/get_started/mnist/beginners.

Google Inc. (2018). TensorFlow: An Open Source Machine Learning Framework for Everyone. Retrieved from: https://www.tensorflow.org/.

Google Inc. (2020a). Google Colaboratory. Retrieved from: https://colab.research.google.com/notebooks/intro.ipynb#scrollTo=5fCEDCU_qrC0.

Google Inc. (2020b). Tensorflow Tutorials. Retrieved from: https://www.tensorflow.org/tutorials/generative/cyclegan.

Granado, J., Vega, M., Pérez, R., Sánchez, J. & Gomez, J. (2006). Using FPGAs to Implement Artificial Neural Networks. *Proceedings of the 13th IEEE International Conference on Electronics, Circuits and Systems*, pp. 934–937.

Guest Blog. (2016). The Evolution and Core Concepts of Deep Learning and Neural Networks. Retrieved from: https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/.

Gwn Lore, K., Reddy, K. K., Giering, M. & Bernal, E. A. (2019, June). Generative Adversarial Networks for Spectral Super-Resolution and Bidirectional RGB-To-Multispectral Mapping. *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR) Workshops*, pp. 926-933.

Hall, R. (2012). *Illumination and color in computer generated imagery*. Springer Science & Business Media.

Han, X., Yu, J., Xue, J. & Sun, W. (2018). Spectral Super-resolution for RGB Images using Class-based BP Neural Networks. *2018 Digital Image Computing: Techniques and Applications (DICTA)*, pp. 1-7.

Hedjam, R., Nafchi, H. Z., Moghaddam, R. F., Kalacska, M. & Cheriet, M. (2015, Aug). ICDAR 2015 contest on MultiSpectral Text Extraction (MS-TEx 2015). *2015 13th International Conference on Document Analysis and Recognition (ICDAR)*, pp. 1181-1185. doi: 10.1109/ICDAR.2015.7333947.

Higgins, I., Matthey, L., Glorot, X., Pal, A., Uria, B., Blundell, C., Mohamed, S. & Lerchner, A. (2016). Early Visual Concept Learning with Unsupervised Deep Learning.

Huang, R., Feng, W., Fan, M., Guo, Q. & Sun, J. (2017). Learning multi-path CNN for mural deterioration detection. *Journal of Ambient Intelligence and Humanized Computing*, 0. doi: 10.1007/s12652-017-0656-4.

Intel Inc. (2018a). Deep Neural Network from Scratch. Retrieved from: https://www.slideshare.net/EranShlomo/deep-learning-from-scratch.

Intel Inc. (2018b). The Specs of Arria 10. Retrieved from: https://www.altera.com/products/fpga/arria-series/arria-10/features.html.

Intel Inc. (2018c). Intel OpenCL Development. Retrieved from: http://www.innovatefpga.com/cgi-bin/innovate/teams.pl?Id=PR029&All=1.

Irick, K., DeBole, M., Narayanan, V. & Gayasen, A. (2008). A Hardware Efficient Support Vector Machine Architecture for FPGA. *2008 16th International Symposium on Field-Programmable Custom Computing Machines.*, (FCCM08.), 304–305.

Isola, P., Zhu, J.-Y., Zhou, T. & Efros, A. A. (2016). Image-to-Image Translation with Conditional Adversarial Networks.

Jaaskelainen, T., Parkkinen, J. & Toyooka, S. (1990). Vector-subspace model for color representation. *J. Opt. Soc. Am. A*, 7(4), 725–730. doi: 10.1364/JOSAA.7.000725.

Kaur, P., Singh, K. & Kaur, H. (2014). Adaptive Modulation of OFDM by using Radial Basis Function Neural Network. *International Journal of Advanced Research in Computer and Communication Engineering (IJARCCE)*, 3(6), 6886–6888.

Kaya, B., Can, Y. B. & Timofte, R. (2018). Towards Spectral Estimation from a Single RGB Image in the Wild.

LeCun, Y., Bottou, L., Bengio, Y. & Haffner, P. (1998). Gradient-based Learning Applied to Document Recognition. *Proceedings of the IEEE*, 86(11), 2278–2324.

LeCun, Yann and Cortes, Corinna and J.C. Burges, Christopher. (2018). The MNIST Database. Retrieved from: http://yann.lecun.com/exdb/mnist/.

Li, J., Wu, C., Song, R., Li, Y. & Liu, F. (2020). AdaptiveWeighted Attention Network with Camera Spectral Sensitivity Prior for Spectral Reconstruction from RGB Images.

Li, Y., Liu, Z., Xu, K., Yu, H. & Ren, F. (2018). A GPU-Outperforming FPGA Accelerator Architecture for Binary Convolutional Neural Networks. *J. Emerg. Technol. Comput. Syst.*, 14(2), 18:1–18:16. doi: 10.1145/3154839.

Liu, X., Ounifi, H. A., Gherbi, A., Lemieux, Y. & Li, W. (2018a). A Hybrid GPU-FPGA-based Computing Platform for Machine Learning. *Procedia Computer Science*, 141, 104 - 111. doi: https://doi.org/10.1016/j.procs.2018.10.155. The 9th International Conference on Emerging Ubiquitous Systems and Pervasive Networks (EUSPN-2018) / The 8th International Conference on Current and Future Trends of Information and Communication Technologies in Healthcare (ICTH-2018) / Affiliated Workshops.

Liu, X., Ounifi, H. A., Gherbi, A., Lemieux, Y. & Li, W. (2018b). A Hybrid GPU-FPGA-based Computing Platform for Machine Learning. *Procedia Computer Science*, 141, 104–111.

Liu, X., Gherbi, A., Li, W. & Cheriet, M. (2019a). Multi Features and Multi-time steps LSTM Based Methodology for Bike Sharing Availability Prediction. *Procedia Computer Science*, 155, 394 - 401. doi: https://doi.org/10.1016/j.procs.2019.08.055. The 16th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2019),The 14th International Conference on Future Networks and Communications (FNC-2019),The 9th International Conference on Sustainable Energy Information Technology.

Liu, X., Ounifi, H.-A., Gherbi, A., Li, W. & Cheriet, M. (2019b). A hybrid GPU-FPGA based design methodology for enhancing machine learning applications performance. *Journal of Ambient Intelligence and Humanized Computing*, 11, 1–15. doi: 10.1007/s12652-019-01357-4.

Liu, X., Gherbi, A., Li, W., Wei, Z. & Cheriet, M. (2021a). TaijiGNN: A New Cycle-Consistent Generative Neural Network for High-Quality Bidirectional Transformation between RGB and Multispectral Domains. *Sensors*, 21(16), x. doi: 10.3390/s21165394.

Liu, X., Gherbi, A., Wei, Z., Li, W. & Cheriet, M. (2021b). Multispectral Image Reconstruction From Color Images Using Enhanced Variational Autoencoder and Generative Adversarial Network. *IEEE Access*, 9, 1666-1679. doi: 10.1109/ACCESS.2020.3047074.

Magnusson, M., Sigurdsson, J., Armannsson, S., Ulfarsson, M., Deborah, H. & Sveinsson, J. (2020, 07). CREATING RGB IMAGES FROM HYPERSPECTRAL IMAGES USING A COLOR MATCHING FUNCTION. *2020 IEEE International Geoscience and Remote Sensing Symposium*.

Makki, I., Younes, R., Francis, C., Bianchi, T. & Zucchetti, M. (2017). A survey of landmine detection using hyperspectral imaging. *ISPRS Journal of Photogrammetry and Remote Sensing*, 124, 40 - 53. doi: https://doi.org/10.1016/j.isprsjprs.2016.12.009.

Maloney, L. T. & Wandell, B. A. (1986). Color constancy: a method for recovering surface spectral reflectance. *J. Opt. Soc. Am. A*, 3(1), 29–33. doi: 10.1364/JOSAA.3.000029.

Marcos, G. F. (2021). Inteligencia Artificial e Aprendizado de Maquina. Retrieved from: https://es.slideshare.net/GeffersonMarcos2/inteligencia-artificial-e-aprendizado-de-mquina.

Martín Abadi et al. (2016). TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems.

Mathieu, M., Couprie, C. & LeCun, Y. (2015). Deep multi-scale video prediction beyond mean square error.

MathWorks Inc. (2019). MathWorks Computer Vision Toolbox / Statistics.

Mitchell, T. M. et al. (1997). Machine learning.

Motamedi, M., Gysel, P., Akella, V. & Ghiasi, S. (2016). Design Space Exploration of FPGA-based Deep Convolutional Neural Networks. *Proceedings of the 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pp. 575–580.

Mukherjee, S. (2017). DARPA Is Spending 65 Million to Fund Brain-Computer Interfaces That Could Cure Diseases. Retrieved from: https://fortune.com/2017/07/10/defense-department-darpa-brain-computer-interface/.

Nagarajan, K., Holland, B., George, A. D., Slatton, K. C. & Lam, H. (2011). Accelerating Machine-learning Algorithms on FPGAs using Pattern-based Decomposition. *Journal of Signal Processing Systems*, 62(1), 43–63.

Ng, A. (2017). Machine learning yearning. *URL: http://www.mlyearning.org/(96)*, 139.

Ng, A. (2018). Machine Learning. Retrieved from: https://www.coursera.org/learn/machine-learning.

Nguyen, R. M., Prasad, D. K. & Brown, M. S. (2014). Training-based spectral reconstruction from a single RGB image. *European Conference on Computer Vision*, pp. 186–201.

Nvidia Inc. (2018). The Specs of TitanXp. Retrieved from: https://www.nvidia.com/en-us/titan/titan-xp/.

Ose, K., Corpetti, T. & Demagistri, L. (2016). 2 - Multispectral Satellite Image Processing. In Baghdadi, N. & Zribi, M. (Eds.), *Optical Remote Sensing of Land Surface* (pp. 57 - 124). Elsevier. doi: https://doi.org/10.1016/B978-1-78548-102-4.50002-8.

Ounifi, H.-A., Liu, X., Gherbi, A., Lemieux, Y. & Li, W. (2018). Model-based Approach to Data Center Design and Power Usage Effectiveness Assessment. *Procedia Computer Science*, 141, 143–150.

Palmer, S. E. (1999). *Vision science: Photons to phenomenology*. MIT press.

Parkkinen, J., Jaaskelainen, T. & Kuittinen, M. (1988, May). Spectral representation of color images. *[1988 Proceedings] 9th International Conference on Pattern Recognition*, pp. 933-935 vol.2. doi: 10.1109/ICPR.1988.28405.

Parkkinen, J. P. S., Hallikainen, J. & Jaaskelainen, T. (1989). Characteristic spectra of Munsell colors. *J. Opt. Soc. Am. A*, 6(2), 318–322. doi: 10.1364/JOSAA.6.000318.

Perera, P., Abavisani, M. & Patel, V. M. (2017). In2I : Unsupervised Multi-Image-to-Image Translation Using Generative Adversarial Networks.

Photography Course. (2021). Multispectral imaging: What is it used for. Retrieved from: https://photographycourse.net/multispectral-imaging-used-for/.

Potluri, S., Fasih, A., Vutukuru, L. K., Al Machot, F. & Kyamakya, K. (2011). CNN based High Performance Computing for Real Time Image Processing on GPU. *2011 Joint 3rd Int'l Workshop on Nonlinear Dynamics and Synchronization (INDS) & 16th Int'l Symposium on Theoretical Electrical Engineering (ISTET)*, pp. 1–7.

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S. et al. (2016). Going Deeper with Embedded Fpga Platform for Convolutional Neural Network. *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, pp. 26–35.

Raina, R., Madhavan, A. & Ng, A. Y. (2009). Large-scale Deep Unsupervised Learning using Graphics Processors. *Proceedings of the 26th annual International Conference on Machine Learning*, pp. 873–880.

Robi Polikar, J. L. S. (2014). General Purpose Processor: Software. Retrieved from: https://www.slideserve.com/aderyn/digital-ii-microprocessors-embedded-systems.

Rokad, B. (2019). Machine Learning Approaches and Its Applications. Retrieved from: https://medium.datadriveninvestor.com/machine-learning-approaches-and-its-applications-7bfbe782f4a8.

Rowe, J. (2017). The Continuing Importance of GPUs For More Than Just Pretty Pictures. Retrieved from: https://www10.mcadcafe.com/blogs/jeffrowe/2017/03/16/the-continuing-importance-of-gpus-for-more-than-just-pretty-pictures/.

Russinovich, M. (2017). Inside Microsoft's FPGA-Based Configurable Cloud. Retrieved from: https://azure.microsoft.com/en-gb/resources/videos/build-2017-inside-the-microsoft-fpga-based-configurable-cloud/.

Samuel, A. (1959). Some Studies in Machine Learning Using the Game of Checkers. *IBM J. Res. Dev.*, 3, 210-229.

Sharp, T. (2008). Implementing Decision Trees and Forests on a GPU. *European conference on computer vision*, pp. 595–608.

Shen, K. (2020). Taijitu. Retrieved from: https://commons.wikimedia.org/wiki/File:Esoteric_Taijitu.svg.

Shi, Z., Chen, C., Xiong, Z., Liu, D. & Wu, F. (2018). HSCNN+: Advanced CNN-Based Hyperspectral Recovery from RGB Images. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1052-10528. doi: 10.1109/CVPRW.2018.00139.

Statistics How To. (2020). What is Root Mean Square Error (RMSE)? Retrieved from: https://www.statisticshowto.datasciencecentral.com/rmse/.

Steinkraus, D., Buck, I. & Y. Simard, P. (2005). Using GPUs for Machine Learning Algorithms. *Proceedings of the 8th International Conference on Document Analysis and Recognition*, pp. 1115–1120.

Stewart, J. W., Vella, J. H., Li, W., Fan, S. & Mikkelsen, M. H. (2020). Ultrafast pyroelectric photodetection with on-chip spectral filters. *Nature materials*, 19(2), 158–162. doi: 10.1038/s41563-019-0538-6.

Stiebel, T., Koppers, S., Seltsam, P. & Merhof, D. (2018). Reconstructing Spectral Images from RGB-Images Using a Convolutional Neural Network. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition Workshops (CVPRW)*, pp. 1061-10615. doi: 10.1109/CVPRW.2018.00140.

Sun, T., Jung, C., Fu, Q. & Han, Q. (2019). NIR to RGB Domain Translation Using Asymmetric Cycle Generative Adversarial Networks. *IEEE Access*, 7, 112459-112469. doi: 10.1109/ACCESS.2019.2933671.

Timothy Lanfear. (2013). High Performance Computing with CUDA and Tesla Hardware. Retrieved from: https://intranet.birmingham.ac.uk/it/teams/infrastructure/research/bear/documents/public/CUDA-2013-07-31/CUDA-Tutorial.pdf.

Tobias Kind. (2018). Tensorflow (TF) benchmarks. Retrieved from: https://github.com/tobigithub/tensorflow-deep-learning/wiki/tf-benchmarks.

Turing, A. M. (2009). Computing Machinery and Intelligence. In Epstein, R., Roberts, G. & Beber, G. (Eds.), *Parsing the Turing Test: Philosophical and Methodological Issues in the Quest for the Thinking Computer* (pp. 23–65). Dordrecht: Springer Netherlands. doi: 10.1007/978-1-4020-6710-5_3.

Verdú, S., Vásquez, F., Grau, R., Ivorra, E., Sánchez, A. J. & Barat, J. M. (2016). Detection of adulterations with different grains in wheat products based on the hyperspectral image technique: The specific cases of flour and bread. *Food Control*, 62, 373 - 380. doi: https://doi.org/10.1016/j.foodcont.2015.11.002.

Vuduc, R., Chandramowlishwaran, A., Choi, J., Guney, M. & Shringarpure, A. (2010). On the limits of GPU acceleration. *Proceedings of the 2nd USENIX conference on Hot topics in parallelism*, 13, 0.

Wang, C., Gong, L., Yu, Q., Li, X., Xie, Y. & Zhou, X. (2017). DLAU: A scalable deep learning accelerator unit on FPGA. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 36(3), 513–517.

Wang, D., An, J. & Xu, K. (2016). PipeCNN: An OpenCL-Based FPGA Accelerator for Large-Scale Convolution Neuron Networks.

Wang, X. & Gupta, A. (2016). Generative image modeling using style and structure adversarial networks. *European conference on computer vision*, pp. 318–335.

Wikia. (2021). CIE 1931 color space. Retrieved from: https://psychology.wikia.org/wiki/CIE_1931_color_space.

FPGA Architecture Presentation. (2009). In *Wikipedia*. Retrieved from: https://www.slideshare.net/omutukuda/presentation-1993175.

Peak signal-to-noise ratio. (2020a). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Peak_signal-to-noise_ratio.

Structural similarity. (2020b). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Structural_similarity.

Taiji (philosophy). (2020c). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Taiji_(philosophy).

Multispectral image. (2020d). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Multispectral_image.

AlphaGo. (2021a). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/AlphaGo.

DeepFace. (2021b). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/DeepFace.

Frank Rosenblatt. (2021c). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Frank_Rosenblatt.

Geoffrey Hinton. (2021d). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Geoffrey_Hinton.

Google Brain. (2021e). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Kinect.

X (company). (2021f). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/X_(company).

Kinect. (2021g). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Kinect.

K-means clustering. (2021h). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/K-means_clustering.

Artificial neural network. (2021i). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Artificial_neural_network.

Machine learning. (2021j). In *Wikipedia*. Retrieved from: https://en.wikipedia.org/wiki/Machine_learning.

Woon Na, Joon Ki Paik & Chul Ho Lee. (1995). An image restoration system for a single-CCD color camcorder. *IEEE Transactions on Consumer Electronics*, 41(3), 563-572. doi: 10.1109/30.468093.

Xiong, Z., Shi, Z., Li, H., Wang, L., Liu, D. & Wu, F. (2017). HSCNN: CNN-Based Hyperspectral Image Recovery from Spectrally Undersampled Projections. *2017 IEEE International Conference on Computer Vision Workshops (ICCVW)*, pp. 518-525. doi: 10.1109/ICCVW.2017.68.

Xu Liu,Abdelouahed Gherbi, M. C. (2019). Boosting Machine Learning Speed with Hardware. Retrieved from: https://substance.etsmtl.ca/en/boosting-machine-learning-speed-hardware.

Yasuma, F., Mitsunaga, T., Iso, D. & Nayar, S. (2008). *Generalized Assorted Pixel Camera: Post-Capture Control of Resolution, Dynamic Range and Spectrum*.

Yasuma, F., Mitsunaga, T., Iso, D. & Nayar, S. K. (2010). Generalized Assorted Pixel Camera: Postcapture Control of Resolution, Dynamic Range, and Spectrum. *IEEE Transactions on Image Processing*, 19(9), 2241-2253. doi: 10.1109/TIP.2010.2046811.

Zhang, C., Sun, G., Fang, Z., Zhou, P., Pan, P. & Cong, J. (2019). Caffeine: Toward Uniformed Representation and Acceleration for Deep Convolutional Neural Networks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(11), 2072-2085. doi: 10.1109/TCAD.2017.2785257.

Zhao, W., Fu, H., Luk, W., Yu, T., Wang, S., Feng, B., Ma, Y. & Yang, G. (2016). F-CNN: An FPGA-based Framework for Training Convolutional Neural Networks. *Proceedings of the IEEE International Conference on Application-Specific Systems, Architectures and Processors*, pp. 107-114.

Zhu, J.-Y., Park, T., Isola, P. & Efros, A. A. (2017). Unpaired Image-to-Image Translation using Cycle-Consistent Adversarial Networks.

Zhu, M., Liu, L., Wang, C. & Xie, Y. (2016). CNNLab: a Novel Parallel Framework for Neural Networks using GPU and FPGA-a Practical Study with Trade-off Analysis. *CoRR*, abs/1606.06234, 0.

Zimbres, R. (2016). Matrix Multiplication in Neural Networks. Retrieved from: https://www.datasciencecentral.com/profiles/blogs/matrix-multiplication-in-neural-networks.