Comparison of Different Deep Learning technologies toward Choosing Appropriate Convolution Neural Network architecture

Yuanjian Tao

Summary

In this project, in order to give guidance before inference CNN models in the different computing environments, I investigate part of popular state-of-the-art CNN model research and source code to find out their similarities and differences. Also, the compression technologies and models are evaluated by the benchmark after model compression.

Though investigating and analysing the benchmark of every model and compression technologies, I find that using dedicated accelerators can significantly increase model efficiency when using more complex models. When the hardware and power support are limited, model compression technologies and compressed models can significantly elevate efficiency. However, after analysing the pruning model results of this project, I found that although pruning can significantly decrease the inference and running time, it may increase the model size and memory footprint unless the pruning rate is higher than a threshold. However, when the pruning rate is too high, the model performance could reduce significantly. About int8 quantization, it can significantly reduce the model size and running time when processing multiple tasks. But only the models which introduce residual learning(He et al., 2016) can gain the inference time decrease. I also try to use two or more compression technologies in the same model, but I found that although different compression methods can implement the same models and gain further size and memory footprint deduction, it cannot bring comparable performance and gain higher efficiency.

Based on the all the results of this project, there is no model with the absolute performance and efficiency advantage. So in order to choose the best models in a specific computing environment, trade-off between different model compression technologies is very important.

List of Abbreviations

EDLS	Embedded Deep Learning System
CNNs	Convolutional neural networks
DNNs	Deep Neural Networks
GConv	Group Convolution
SWaP	low size, weight, and power
FPGA	Field-programmable gate array
ILSVRC2012	ImageNet Large Scale Visual Recognition Challenge 2012
GFLOPs	Giga Floating-point Operations Per Second
CUDA	Compute Unified Device Architecture
BN	Batch Normalizing
ReLU	Rectified Linear Unit
AM	Arithmetic Mean
GM	Geometric Mean
ARC3	Advanced Research Computing3
ARC4	Advanced Research Computing4

Table of Contents

List of Figures

List of Table

Chapter 1
Introduction

1.1 Project Aim

Convolution Neural Networks(CNNs) have been used in different fields in recent years. Benefiting from performance elevation, the general devices such as smartphones and CCTV video recorders also contain more intelligent performance when using CNNs. However, with the performance improvement of CNNs, the required computing power also increases. Restricted by limited power and the size of devices, the majority of devices cannot reach the computing power demand of state-of-the-art CNNs. Therefore, different compression technologies and models are proposed. So it is very important to review them before choosing and using a CNN model.

In this project, different cutting edge CNN models and compression technologies will be compared in vertical and horizontal angles. Meanwhile, different properties and benchmarks of models will be evaluated to extract the crucial characteristics. The project results should provide accurate model performance and efficiency. Also, different compression technologies are analyzed and then present suggestions when using them.

1.2 Objectives

This project will meet the objectives and obtain the results are following:

1. Understanding the state-of-the-art CNNs and compressed models structure. And comparing their similarities and differences.
2. Acquiring the models which are worthy of study. Using model compression technologies such as pruning and quantization to compress CNN models. Logging and comparing the basic differentiation between compressed and uncompressed models.
3. Using different hardware and evaluation methods to benchmark all the generated models and noting all the useful data in the database.
4. Processing the raw data and evaluating all the models performance and efficiency. Giving detailed and accurate test results.
5. Based on benchmark and Investigation results, giving my comments and recommendation when using different CNN architecture and compression technologies.

1.3 Deliverables

This project has the deliverables bellowing:

i. The MSc project report
ii. All the codes that have been used include pre-trained models download, model transform, model testing, and model analysis.
iii. All the recorded raw data contain different servers testing logs, distributed testing model records, and model base information screenshot records.
iv. Analysing raw records to generate the extracted data which include model basic information, average performance, confidence interval, and models rank.
v. Statistic figures and its generating code.

vi. A GitTea repository that contains the source code, records, analytics results, and statistical figures.(see Appendix A.3)

1.4 Ethical

No related ethical issue occurred related to my project.

Chapter 2
Background research

2.1 Literature Survey

This chapter discusses the research and technology about CNNs and compressed technologies of CNNs. It explains why CNNs need to be compressed and previous research about how to compress CNNs.

Along with the development of CNNs, in order to achieve higher performance, CNN models become larger which demands more calculational electrical power. Meanwhile, they own better performance which attracts various devices, especially embedded and mobile devices to use them. However, state-of-the-art DNNs often are trained in a distributed computing environment with dedicated chip which can provide significant computational power. Also, the models occupy more storage and cannot easily run on a limited power platform. Hence, a trade-off between model preference and efficiency is very necessary in particular cases.

In order to solve these issues, many algorithms and fine-tuning methods are emerging such as pruning, quantization, and compressed models.

2.1.1 Convolution Neural Networks(CNNs)

Along with CNNs development, models become larger which brings better accuracy. Meanwhile, higher computing power and hardware demand also increase CNNs threshold.
In 1968, Hubel and Wiesel found that brain context has a very spatial arrangement that can find the objective features and classify them. (Hubel and Wiesel, 1968) Based on their theory, many researchers try to simulate this behavior which calls convolution operation. In 1998, LeCun(LeCun et al., 1998) successfully simulated this phenomenon and construct first CNN which had significant performance when implementing it in the image classification. Because of hardware and computing power restriction, in this research, LeCun *et al.* implemented a 7 layer network which used 2 convolution layers, 2 maxpooling, 2 fully connection, and one Gaussian connection. Along with hardware performance, algorithm, and computing power increases, in 2014, AlexNet was introduced which could be trained by muti-GPU. Meanwhile, it added dropout, data augmentation, and local response normalization.(Krizhevsky, 2014) In the same year, VGG(Simonyan and Zisserman, 2014) had deeper layers and smaller convolutional kernels which proved that raising the network depth can significantly enhance accuracy. In GoogleNet, inception architecture uses optimal local sparse structure in a convolutional vision network. (Szegedy et al., 2015) In order to reduce degradation and train deeper models in DNN, residual learning was introduced which can significantly raise the network layers and help to increase the accuracy. (He et al., 2016) In the research of Xie et al., residual learning and inception theories are mixed and used in ResNeXt although it cannot offer very good performance. (Xie et al., 2017)

As the Table 2.1 illustrates that along with CNNs development, networks can be built with deeper layers and consider more specific cases. Hence, the parameters and size of CNNs become larger which need more training time and implemented power.

Table 2.1 Compare different CNN size and implement cost

Model Architecture	Year	Total size (MB)	Parameter number	Giga Floating-point Operations Per Second(GFLOPs)
LeNet	1998	25.68	5,612,426	0.04
AlexNet	2014	233.96	716,084,224	0.72
VGG11	2014	632.78	132,863,336	7.63
VGG11_bn	2014	506.85	132,868,840	7.65
VGG19	2014	787.31	143,667,240	19.67
VGG19_bn	2014	900.66	143,678,248	19.7
GoogleNet_inception	2015	144.43	13,004,888	1.52
ResNet18	2016	279.2	60,192,808	11.62
ResNet152	2016	279.2	60,192,808	11.62
ResNeXt50_32x4 dimension	2017	95.7	25,028,904	4.29
ResNeXt101_32x8dimension	2017	339	88,791,336	16.55

2.1.2 Inference cost in Embedded Deep Learning System(EDLS)

In the EDLS, there are heterogeneous processors and accelerators to implement DNNs. They have different operational instruction sets, performance, and application scenarios. However, in order to raise the training efficiency and model performance, muti-chip assessors, or distributed systems often are introduced in the training. There is a different performance gap between training and running systems. Therefore, cutting the parameter number and transforming models to adapt low size, weight, and power (SWaP) accelerators is very necessary.

Reuther *et al.*,(Reuther et al., 2019) try to evaluate the Machine Learning accelerators. (as Figure 2.1 shown) In this research, they found that a majority of low-SWaP processors and accelerators only support inference lower Integer computation precision. A very limited number of low-SWaP devices and accelerators (such MIT Eyeriss and TPUEdge) can implement higher computation precision. However, they cannot offer comparable performance and reasonable power performance. In the embedded chips and systems set, although they already increase the computation precision, they have almost the same Giga operations per second (GOps/s) as very low power and research chips.

Compared with low-SWaP accelerators, data center chips and cards have high computation precision and significant GOps/s in single-chip and card. In state-of-the-art DNNs training, they can be implemented by distributed system or service system which provides more performance in the training and generation. For instance, according to Sanh *et al.*, model parameter number have rapid growth which brings significant improvement in Natural Language Processing (NLP). (Sanh et al., 2019) Such BERT-Large needs a DGX-2 server with 16 V100 cards which need under 3 day to train its model. If using a distributed 16 DGX-2H nodes system with V100 cards, its training time can be shortened to 236 min. (Narasimhan) Also, inference the high parameter model consumes significant hardware resources. For example, using 1080Ti, which has the same core as P100 needs 5.2 milliseconds to inference BERT-Large one time(Yang et al., 2020). Therefore, training and inference original high parameter number models demand high-performance accelerators.

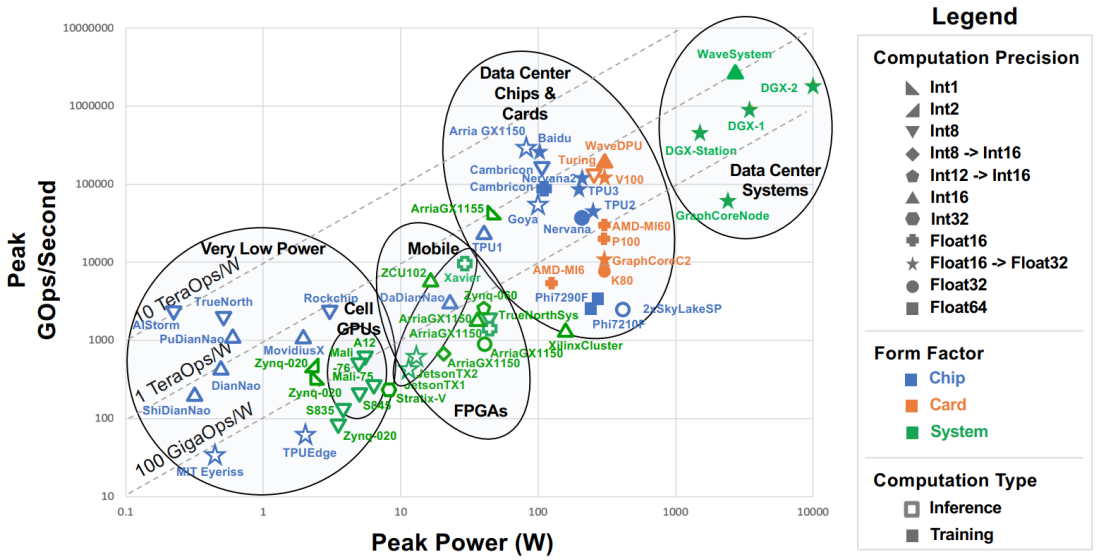


Figure 2.1 Performance vs. power scatter plot of publicly announced AI accelerators and processors([Reuther et al., 2019](#))

In conclusion, different power support brings different architecture and operational characteristics between high and low power accelerators. In order to introduce DNNs in EDLS, transforming the lower computation precision parameter, and cutting the parameter number is very necessary.

2.1.3 Pruning

In order to implement state-of-the-art models in the low-SWaP hardware, pruning some weights and perceptrons to slim network is one of the research directions which not only can maintain the performance of models but also decrease the demand of hardware.

In the research of CNN architecture, Denil et al. found that some of perceptrons and weights have low accesses which are redundancies and should be removed. Another result is that some perceptrons and weights need to not be learned at all. (Denil et al., 2013) Another research by Hu et al. shows that after Rectified Linear Unit(ReLU), VGG-16 has a very high percentage of zero activations of a neuron. (Hu et al., 2016) In the research of Srinivas and Babu, CNNs contain some redundant neurons which cannot increase accuracy meanwhile add unnecessary calculation, memory, and storage. (Srinivas and Babu, 2015) Therefore, pruning such weights and perceptrons can achieve the slim target of models. Research of Denton *et al.* tries to use pruning, (as Figure 2.2 shows) that through compression and fine-tuning, all the layers of performance are restored. (Denton et al., 2014) As a result, we can use pruning to remove those weights and perceptrons which not only can decrease meaningless calculation and memory footprint, but also can increase CNNs performance in specific cases.

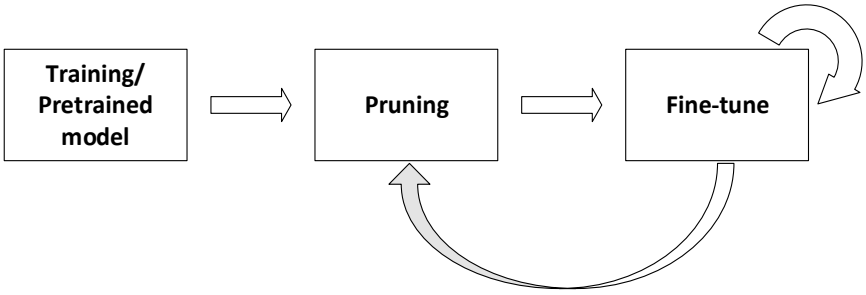


Figure 2.2 Pruning process

In partial training, Haeffele and Vidal(Haeffele and Vidal, 2017) have proven that when networks have large enough parameters, they may have few or one global minimizer. This means that training large networks can be easier than smaller networks. Therefore, compared with designing and training a small network, training, pruning, and fine-tuning a larger network should have higher efficiency in small network training.(Frankle and Carbin, 2018)

2.1.4 Quantization

Quantization is another method that can achieve a balance between performance and efficiency. According to Gong *et al.*, using weight or conducting product quantization can significantly decrease model size. But it only loses classification accuracy in specific CNNs.(Gong et al., 2014)

High performance CNNs often trained by distributed or multi-GPU computing environment. Because it can accelerate the training efficiency. Also, the models can be designed deeper and larger. However, this also make the margin between training environment and implemented devices become bigger. According to the research of Dundar, the performance of training environment is dozens of times than mobile devices.(Dundar et al., 2013) Meanwhile, low price and power consumption chips such as X86 and ARM CPU cannot have very high efficiency when running float32 models. If the embedded system chooses to allocate enough memory, storage, and dedicated calculational chip such as ASIC, FPGA, or GPU to run and speed up CNNs, it could significantly increase budget and power consumption.

Quantization neural networks(QNNs) is one of theory which try to solve this issue. Its thought is convert the higher precision operators and weights to be lower. Although it loses some precision, it can significant reduce models size, memory footprint, and increase running efficiency. Meanwhile, quantized operators can be easily converted to simple and pre-existing operators which can be easier implemented in the X86 or ARM architecture chips. (Jain et al., 2020) As a result, it can reduce hardware demand and is beneficial for low-SWaP devices.

2.1.5 MobileNet

Both pruning and quantization are based on the existing model architecture. They also have some limitations and constraints in the model compression. Some researchers have concluded that using dedicated algorithms or cutting the network layers also can achieve the same objective. Therefore, some state-of-the-art CNNs already consider preference and efficiency when they are designed like MobileNet.

In 2017, MobileNet was proposed. Benefiting from Depthwise Separable Convolution to replace traditional convolutional algorithms and Width Multiplier, it achieves considerable efficiency and usable accuracy. (Howard et al., 2017)

In the MobileNetv1, the architecture of MobileNet is a direct connection from input to output. Its architecture is similar to VGG. In convolutional calculation, MobileNet is separated into two steps which are 3x3 depthwise convolution and 1x1 convolution. (Figure 2.3 and Table 2.2) Because depthwise convolution does not combine all the filters output to generate new feature maps. Computational computation of Depthwise Convolution has 8 to 9 times less than traditional convolution. Beneficial from width multiplier α , MobileNet can adjust output channel and the parameter number which can reduce running time and model size.

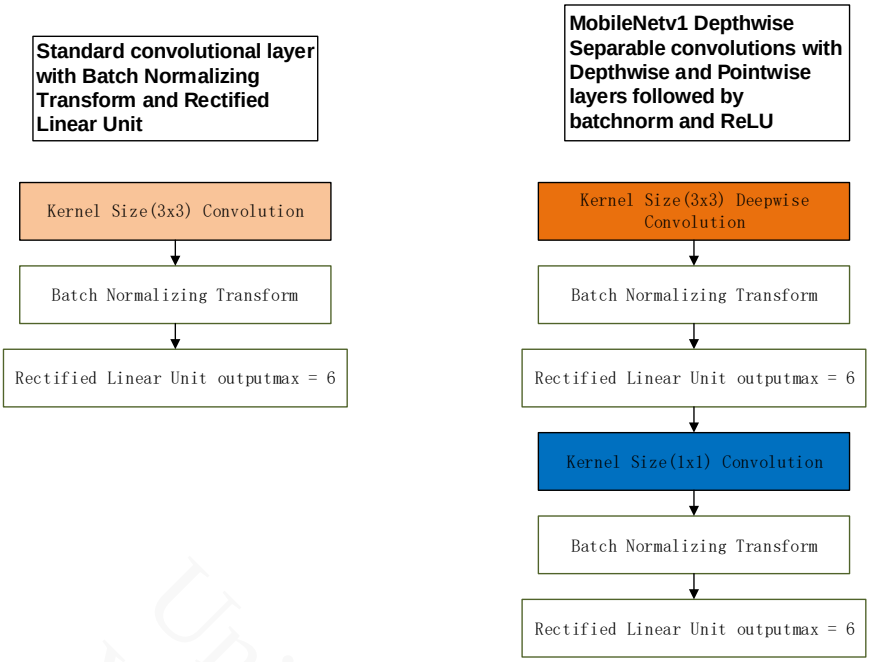


Figure 2.3 Standard and Depthwise Separable convolution with Batch Normalizing(BN) Transform and ReLU.

Table 2.2 MobileNetv1 network layer structure and parameters

Block type	Stride	First convolution kernel size	Input channel	Output channel
Standard Convolution with BN	2	3x3	3	32
Depthwise Separable Convolution	1	3x3	32	64
Depthwise Separable Convolution	2	3x3	64	128
Depthwise Separable Convolution	1	3x3	128	128
Depthwise Separable Convolution	2	3x3	128	256
Depthwise Separable Convolution	1	3x3	256	256
Depthwise Separable Convolution	2	3x3	256	512
Depthwise Separable Convolution	1	3x3	512	512
Depthwise Separable Convolution	1	3x3	512	512
Depthwise Separable Convolution	1	3x3	512	512
Depthwise Separable Convolution	1	3x3	512	512
Depthwise Separable Convolution	1	3x3	512	512
Depthwise Separable Convolution	2	3x3	512	1024
Depthwise Separable Convolution	1	3x3	1024	1024
Average Pool	0	kernel size = 7x7		
Fully connective				
Softmax				

MobileNetv2 (Sandler et al., 2018) not only continually inherits MobileNetv1 successful characteristic, but it introduces new methods such as linear bottlenecks and inverted residuals which is similar to ResNet architecture. (See Table 2.3 and Table 2.3) They further increase model accuracy and further decrease model size and hardware demand(As Table 2.4).

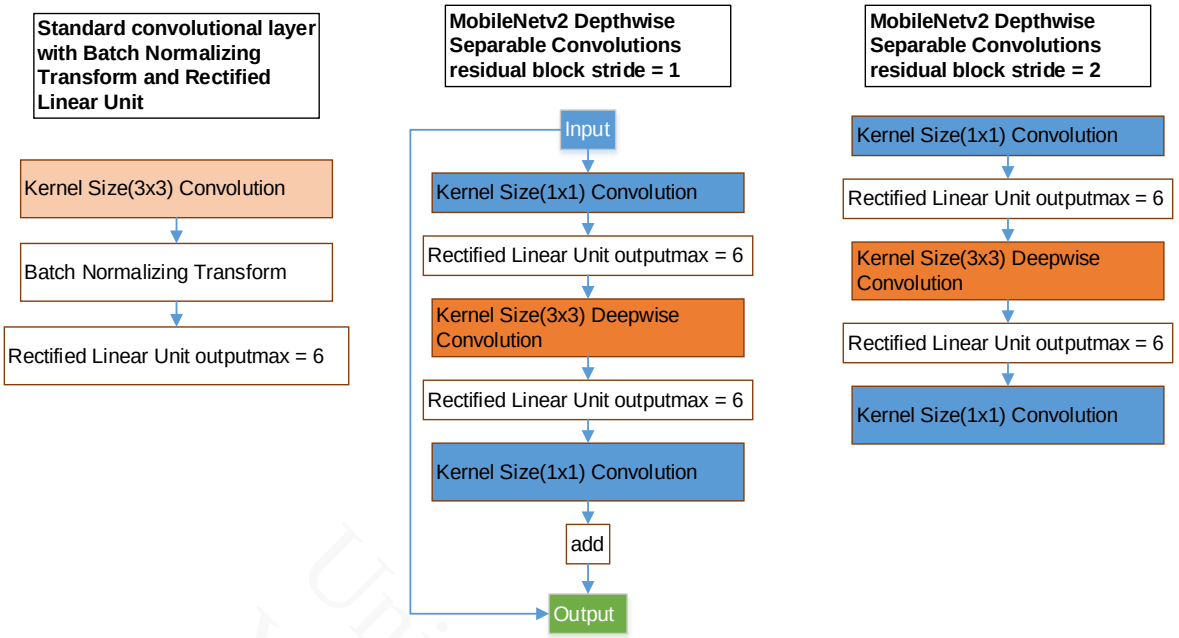


Figure 2.4 MobileNetv2 Standard Convolution with BN, stride = 1 and stride = 2 Convolutional blocks

Table 2.3 MobileNetv2 Body Architecture

Block type	Stride	Input channel	Output channel	expansion factor (t)	repeat number (n)	other
Standard Convolution with BN	2	3	32			Convolution kernel size = 3x3
Residual block	1	32	16	6	1	
Residual block	2	16	24	6	2	
Residual block	2	24	32	6	3	
Residual block	2	32	64	6	4	
Residual block	1	64	96	6	3	
Residual block	2	96	160	6	3	
Residual block	1	160	320	6	1	
Standard Convolution with BN	1	320	1280		1	Convolution kernel size = 1x1
Avgpool						kernel size = 7x7
Fully connective		1280	1000			
Softmax						

Table 2.4 Compare MobileNet v1 and v2. (figures in MobileNet v1 Top-1 are cited from Sandler (**Sandler et al., 2018**))

Model Architecture	Total size (MB)	Parameter number	GFLOPs	Top-1 accuracy
MobileNet v1	176.56	4,231,976	0.58	70.6%
MobileNet v2	166.81	3,504,872	0.31	71.9%

2.1.6 ShuffleNet

MobileNetv1 introduces some features about previous model theories which bring significant promotion in performance per watt. Therefore, just as the ResNeXt, ShuffleNetv1 tries to introduce more state-of-the-art architecture in order to achieve better performance.

Compared with MobileNetv1, ShuffleNetv1(Zhang et al., 2018) introduces similar ResNet architecture(He et al., 2016) as earlier. Each ShuffleNetv1 also introduces 3x3 Depthwise separated convolution instead of 3x3 convolution in each block. to sample the feature maps. Likewise, Depthwise separated convolution also needs to use 1x1 convolution to sampling the feature maps to lift or reduce dimensions of feature maps. But in the ShuffleNetv1, 1x1 convolution is replaced by 1x1 group convolution(GConv).(See Figure 2.6 middle)

Benefiting GConv, each input channels cannot be affected by another channels. Although it can significantly reduce model parameters in GConv, it cannot take another channel's feature maps into account either so it could reduce the model performance. Therefore, ShuffleNetv1 introduces channel shuffle layers(See Figure 2.5) which can allocate channel pieces to each convolutional group. (Zhang et al., 2018) It makes each output from each GConv consider all the channels. Besides, it decreases calculational consumption.

After final convolution, in order to avoid residual is zero, ShuffleNetv1 also needs to introduce the result from the previous block. Compared with ResNet(He et al., 2016), it uses an average pooling and concats function to sample and add all the cases to the final result. (See Figure 2.6 right)

Figure 2.5 Channel shuffle layer.

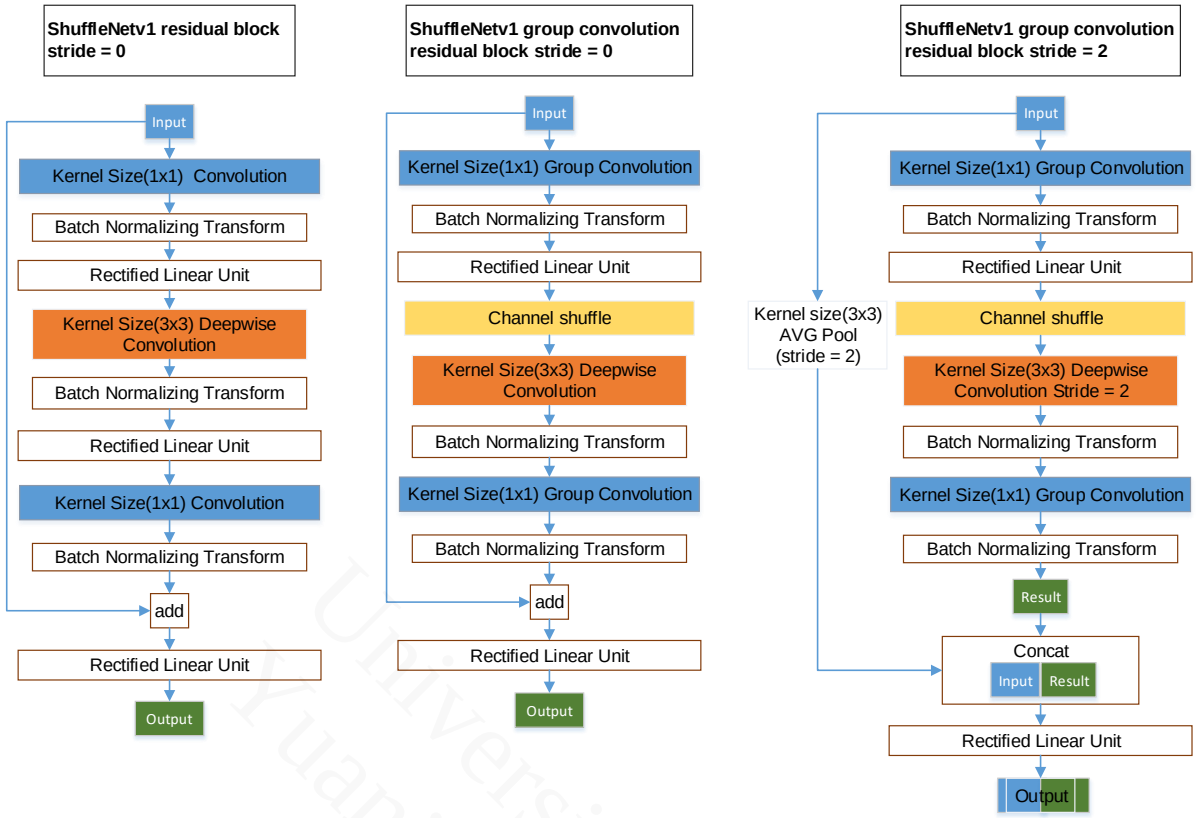


Figure 2.6 ShuffleNetv1 residual convolution, stride = 1 GConv, and stride = 2 GConv blocks.

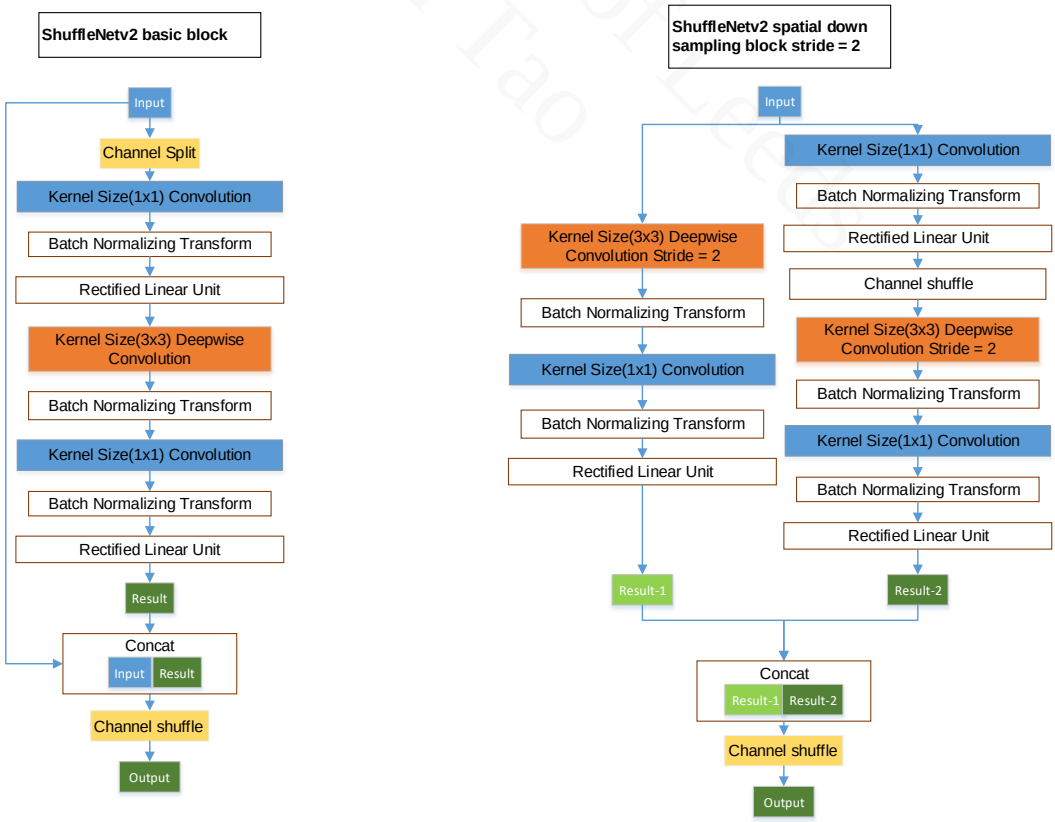


Figure 2.7 ShuffleNetv2 basic block and spatial down sampling block.

In ShuffleNetv2(Ma et al., 2018), GConv is abandoned and replaced by channel split in down sampling block (As shown in Figure 2.7left) According to Ma, it can increase double output channel as ShuffleNetv1. Its architecture can upkeep the parameter number and elevate its accuracy.

2.1.7 Conclusion

This chapter illustrates that CNNs derive from bionics which simulates the cerebral cortex. Along with development of calculational power and algorithms, CNNs have significant performance and reach some achievements in image classification and segmentation. In order to gain higher accuracy and better performance, CNNs become deeper. This brings more parameters and high calculational demand. But in practice, such as automatic drive and real-time image/video analysis, they have very limited hardware and need to trade-off the performance and calculational consumption. Therefore, model compression technologies are very important when implementing DNNs in EDLS. Each compression technology has different characteristics. In order to design efficient EDLS, we need to know behaviour to understand when using different compression technologies. (Qin et al., 2018) It can help developers to construct approximate deep learning systems in embedded devices.

2.2 Methods and Techniques

In order to test and analysis all the cutting edge models and compression technologies, investigating different deep learning frameworks, implementable CNNs, determining which pruning and quantization methods can be review, and appropriate dataset are very import. In this section, some different methods and technologies which contribute to this project are discussed and trade-off their strength and drawbacks.

2.2.1 Deep Learning Framework

In order to implement state-of-the-art CNNs and compressed technologies, choosing a clear, tidy, and efficient deep learning framework is very important and could significantly affect development efficiency and available resources.

TensorFlow is one of the early examples of architecture in Machine Learning. Because it has been used for so long, it has more documents, third-part tools, pre-trained models, and various open-source resources. It already is implemented by variety of heterogeneous hardware systems such as Embedded and high-performance computers. (Abadi et al., 2016) As a result, in industrial manufacturing, it is very popular.

Pytorch(Paszke et al., 2017) is one of the popular frameworks which based on Torch architecture and deeply utilizes Python programming style. It not only has as efficient C++ core as TensorFlow but also it owns simplified and clear code structure to run automatic differentiation and weight update. According to Paszke, Pytorch is based on Torch and utilizes Python program style which brings better usability and reduces learning time cost. Benefiting from these advantages, it is accepted by scientific researchers and primary Machine Learning beginners. Plus, Python also provides interoperability and extensibility which can easily implement and transform other open-source libraries like Numpy. Clear class structure also helps the users locate crucial code blocks and increase the usability. Finally, muti-core and multi thread support offer GPU adaptation and higher efficiency. (Paszke et al., 2019) Such Machine Learning framework could significantly increase the efficiency of development; so it is a very powerful tool in scientific research and education.

Because the majority of Deep Learning Framework can support various programming languages, the programming language selection is very important. The development of programming languages is accompanied by the development of computers. Early languages such as assembly and C have good operating efficiency. Along with computer architecture development, programming languages not only have acceptable efficiency but also should be easier to read and improve team programming efficiency. Object-oriented programming languages such as C++ and Java have come out.

After the millennium, programming languages are used in different aspects which demand language that should be easier to learn and more similar to natural language. Therefore, Python and GO have become popular.

Benefit from rich API support, although TensorFlow is based on C++, it can be dispatched by other programming languages such as Python, Java, and GO.(Abadi et al., 2017) Compared with it, because Pytorch is based on Python programming style, it only can support the target language Python and its underlying language C++.

2.2.2 Pruning

In CNNs pruning, the most general method is to continually prune the network and fine-tune. So which weights and perceptrons should be pruned and, need to be pruned is a crucial problem.

Several researches use different methods to choose and cut the weight and perceptrons in pruning architecture. To begin with, Lin et al. introduce L1-norm based channel pruning. (Li et al., 2016) Its method though counts each perceptron weights and sorts than to find which perceptron is useless. Moreover, except for weights, the parameter of Batch Normalizing(BN) also can be used to decide which perceptron should be cut. According to Liu et al., as Figure 2.8 BN also needs to be trained in CNNs training γ and β to limit the output in the finite field. This means that if γ is a scaling factor which directory signs that which perceptrons are important.(Liu et al., 2017) Therefore, through estimating BN parameters, the specific perceptron can be decided whether prune it or not. Moreover, the feature map also can be used to decide which perceptron should be pruned. In the research of Hu et al., After ReLU Unit, much lower value output could become zero. (Hu et al., 2016) So if some feature map contains too much zero output. This means that the filters which generate their features is not very important. Finally, in order to not lose the information when cutting the perceptrons and weight. Lin et al. uses Filter Sketch to describe the important information in the CNN and compare the original and pruned model covariance to reduce information loss. (Lin et al., 2020) This can avoid time consumption as another method to recursively prune and fine-tune the network.

$$\hat{z} = \frac{z_{in} - \mu_{\beta}}{\sqrt{\sigma_{\beta}^2 + \epsilon}} \; ; \; z_{out} = \gamma \hat{z} + \beta$$

Figure 2.8 Equation of BN(**Liu et al., 2017**)

2.2.3 Quantization

In order to upkeep the performance of models and learning ability, quantization also has many different methods to transform and fine-tune the model such as dynamic quantization and static quantization. Quantization also can utilize different storage types of weight such as int32, int16, int8, and int4.

In general, each pixel from images is saved by float32. Hence, CNNs architecture also uses float32 to save the weigh. This architecture can reduce the logic complexity of calculation

and upkeep precision of the operators and weight. In order to raise the calculation efficiency in x86 CPU, ARM, and specific Field-programmable gate array(FPGA), quantization technology transforms the weights storage type from float32 to the lower precision type like int8. But if calculating float32 operators with int8, int8 has to raise the precision from int8 to float32. This does not accelerate the calculation time. Therefore, the image needs to reduce the precision from float32 to int8. (Jacob et al., 2018) In partial, quantized models need to add quantize and dequantize functions that wrap to pre-process float-valued input and output.

After the pre-processing, the model can be generated by dynamic quantization or static quantization. In dynamic quantization, weights are saved by lower precision type and initialize at random. The dataset and training method are the same as the unquantized models. However, it cannot utilize GPU or other float training accelerators to increase the training efficiency. As for static quantization, a pre-trained model which is trained by float32 should be provided. The quantization process needs to transform itself from float32 to a lower precision type like int8 and ignore the precision loss. It also can be continually fine-tuned in order to recover some accuracy.

2.2.4 CNNs pre-trained models

State-of-the-art CNNs often have enormous parameter numbers which demands large amounts of computing power to train them. They also need fine-tuning to ensure the best performance as possible. So if training all the models from random weights, it could take a very long time. In order to decrease the time of development and improve the rate of reuse code, TensorFlow and Pytorch community also provide some pre-trained models which can be implemented straightforwardly. But they are also trained by specific configuration so in the practical case, they need some necessary modification and fine-tuning.

2.2.5 Testing Dataset

Each CNN architecture model is trained on different datasets which could train CNNs to get different functions. Therefore, the dataset also is a very important aspect of CNNs.

In general, CIFAR-10 and CIFAR-100 dataset(Krizhevsky and Hinton, 2009) have 10 and 100 classifications, 60 thousand images, and 32x32 image resolution(as Figure 2.8 shown) which are often used in CNN models test and benchmark. Because of lower data capacity, the majority of the Machine Learning community does not provide pre-trained models. Besides, because of lower image resolution and classification, it cannot be used in partial applications.

Figure 2.9 CIFAR-10 image sample

Another popular dataset is ImageNet dataset. (Russakovsky et al., 2015) It utilizes full resolution images and now reaches up to 14 million images and 21 thousand synsets indexed. It not only can be used to evaluate the model performance but also train models to use in practical application. Benefiting from high resolution(as Figure 2.10 shown) and total images number, models can learn more features and get more detail from images. Therefore, ImageNet test results can be closer to actual test results. Also, it can detect very subtle differences between models. But using it to train and test the models could consume more time and computing power.

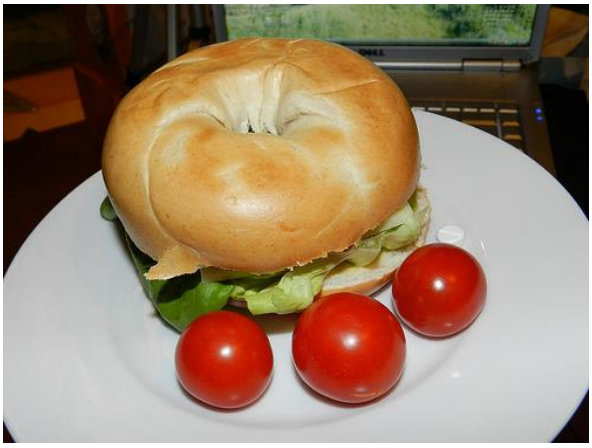


Figure 2.10 ImageNet Large Scale Visual Recognition Challenge 2012(ILSVRC2012) validset class07693725 00044186 image sample

2.3 Choice of methods

In section 2.2, many different tools and technologies are discussed that have different characteristics. In order to use available resources to benchmark different CNN architecture and compressed technologies, choosing appropriate and avalanche technologies is very necessary. In this section, different technologies will be compared and considered in specific cases to choose which technologies are more suitable and efficient.

In this project, analysing model structure and extracting the log and information is very important jobs. Pytorch can be qualified in this project. Meanwhile, Pytorch also can be allocated in University of Leeds Advanced Research Computing(ARC) clusters. ARC can provide state-of-the-art V100 GPU which can significantly increase the testing efficiency and giving the information of cutting edge hardware performance when running CNN. In order to better use Pytorch, Python is chosen to be the programming language in this project. Using Python also has many benefits. For example, its language style is similar to native language. Also, it has rich documents of development can be referred. This can be beneficial for this project.

Regarding pruning, based on the majority of pruning structure, pre-trained models demand repeated pruning and fine-tuning to recover the accuracy of networks. Because of the limited time and computing power of ARC3, pre-trained, pre-pruned, and pre-fine-tuned models have to be chosen in this project. In the research of Lin et al., (Lin et al., 2020) these models already are provided in GitHub although the model architecture only can choose ResNet50. But, they also provided different compressed rates which could be evaluated.

Quantization of CNNs often just needs to transform pre-trained models which save the weight by float32 to lower precision operators. According to the Pytorch online document,(Contributors) Pytorch only supports 8-bit weights and activations in Neural Network, but it does not limit CNN architecture to do quantization. Therefore, VGG, ResNet, inceptionv3, MobileNetv2, and ShuffleNetv2 will be quantized and tested in this project.

In Pytorch, TorchHub library(Facebook) can automatically implement the different CNNs architecture structure. Also, it provides the pre-trained parameter which can download pre-trained models straightforward which include quantized and unquantized models. As a result, unquantized pre-trained models will be downloaded in it. Its pre-trained models are transformed into quantized models and tested as the benchmark.

About testing dataset, compared with CIFAR-10 and CIFAR-100, ImageNet can test models more precisely. This can be beneficial for data analysis. Meanwhile, Pytorch also has many pre-trained models that are trained by ILSVRC2012. This can reduce the time consumption in training.

In conclusion, consider the trade-off between particular time, resource, and technical limit. I will use Pytorch to download the pre-trained float32 models from TorchVision, use Pytorch interface to generate int8 quantized models, and implement ILSVRC2012 dataset to test all the models.

Chapter 3
Datasets and Experimental Design

3.1 Datasets/Data Sources

If a model is efficient, it means that a model not only has acceptable accuracy but also the model size and demand computing power can be small as small possible. Therefore, basic research directions are performance, computing power, and storage.

Indicating a model's performance needs to have many different indicators. The most basic indicators are Top-1 and Top-5 accuracy. Meanwhile, precision, recall, and F1 score(Sasaki, 2007) also can evaluate the preference of model abilities in the classification. Finally, the cross-entropy loss can show the inference precision between model and result.

If the models need to evaluate the computing power consumption, when the weight type is determined, the model parameter number can indicate the model calculus consumption and calculate the GFLOPs. However, when the models are quantized, the weight store type is changed, so the test hardware can be determined. The models run quantitative data in the same condition and compare the execution time.

Finally, storage occupancy, memory, and disk usage need to be considered together. Additionally, when the models implement Compute Unified Device Architecture (CUDA)(Tölke, 2010), the CUDA memory footprint needs to be considered.

3.2 Experimental Design

3.2.1 Test data collection and allocation

In chapter 2, ImageNet dataset is chosen when testing the models. So the first step to designing the experiment is to download and deploy ImageNet data. But all the ILSVRC2012 training, valid, and test dataset need to download 158 GigaBit(GB). It also needs a very long time to test all the data and the pre-trained models are trained by ILSVRC2012 training set. Therefore, ILSVRC2012 valid set which only has 6.4 GB and 50 thousand image files and is testable in ARC3 is chosen in the model benchmark. But when comparing quantized and unquantized models efficiency, CUDA accelerators cannot implement quantized models. So the unquantized model needs to be tested by CPU whose testing time consumption is very high. Therefore, the efficiency testing dataset will use one thousand images that are taken from ILSVRC2012 testing dataset to test the overall running time when implementing all models.

3.2.2 Prepare trained, pruned, and quantized models

Pre-trained models are downloaded from two different sources. CNN pre-trained models can be downloaded by TorchVision(Facebook) directly and save their state dictionary. As for pre-pruned models, using the open-source code which was created by the Lin et al.,(Lin et al., 2020) the pre-pruned models could be downloaded on GitHub and implement.

Now onto Pytorch static quantization guidance. (Raghuraman Krishnamoorthi) To begin with, Convolution, BN, and ReLU are fused by fuse model function which also needs to be defined by each layer or specific logic. This process fuses the three-layers and quantizes all the

parameters together which can prevent the parameters from being independent and decrease the accuracy. Meanwhile, the float32 model needs to be warped by QuantStub and DeQuantStub functions(see Figure 3.1) so that they can convert between float32 and int8 tensors. After preparing quantizable models, quantization configuration needs to be defined; containing which type of weights will be converted and what device will use this model. Benefiting from Pytorch structure, using Pytorch API can convert float32 to int8 models and implement them by C++ API. Finally, because quantized models are needed to transfer to C++ API to implement before saving the model, the models need to be serialized.

Figure 3.1 quantized model dataflow.

3.2.3 Test design

Because different model types have different structures that cannot use the same method to detect basic model information. Using TorchSummary(Chandel) and Pthflops(Bulat) open-sourced analysis tools, the float32 model parameter number, estimated model size, and GFLOPs are estimated.

In order to compare all types of models horizontally, experiments can be designed to allow all models to complete the same tasks under the same hardware conditions. As the dataset design, 5,000 images that derive from ILSVRC2012 testing dataset are input into each model. Besides, the models are all run in one ARC3 standard node(Dixon, 2016) separately. Its hardware is Intel “Broadwell” 2x E5-2650v4 2.2GHz 24 cores CPU and 128GB of memory which is run on CentOS Linux release 7.6.1810(CentOS, 2019). Also, all the parameters in the code include batch size and image resizes are the same. These tests are run one hundred times.

To test and compare each float32 pre-trained model performance such as accuracy and precision, all the ImageNet validset is run by all models. So utilizing accelerators like GPU to speed up the tests is very necessary. In this project, ARC4 NVIDIA V100 GPU is used to test these models. However, quantized models cannot use GPU to accelerate the tests. But the quantized models have higher efficiency when using CPU so it does not consume too much time. All the tests are run 20 times. Each test will collect the inference time before the test which time one image classification process.

3.2.4 Data analysis

In order to increase the test accuracy and avoid the noisy data which generate by unpredictable reasons such as electrical support and cluster temperature increase to affect the final results, all the tests are recursively run many times to generate some result sets. All the results are average by Arithmetic Mean(AM) and Geometric Mean(GM). Meanwhile too high or low results are eliminated and getting the 90% confidence interval.

All the collected data will be analysed from vertical and horizontal. Horizontal analysis is mean that all the models will be tested by the same environment. Include the same hardware, dataset, and parameters. Vertical analysis demands the same CNN model is tested by different hardware or compression technologies. This could evaluate the characters from different angles.

Not only the raw data but the analysis data could be saved in the dataset. The entire analysis process and method could be finished by Python scripts and written down into the dataset. This can significantly increase the efficiency of data analysis. Also, the process

could be monitored and checked. When the raw data changes or some part of the analysis process needs to be changed, all the analysis results and figures could be regenerated by the scripts.

In conclusion, although the analytical method and process are complex, using Python to analyse and generate visual figures can significantly increase the efficiency in the analysis. Meanwhile, all the processes can be checked and changed by Python scripts; this can elevate the review efficiency. Also, this can increase data accuracy.

Chapter 4
Results of the Empirical Investigation

4.1 Pre-trained float32 models comparison

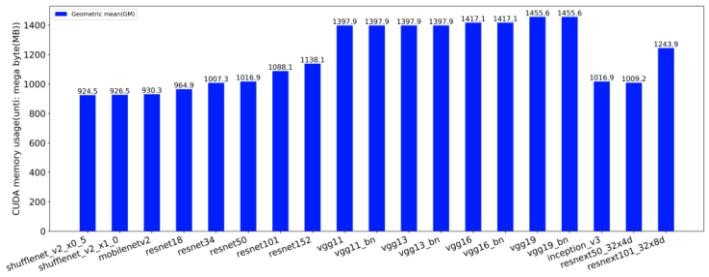
In this test, all the float32 pre-trained models are tested by one V100 GPU accelerator which is installed in Advanced Research Computing4(ARC4) 40core-192G server. Some of the experiments are executed by one ARC3 normal computing 24core-128G node.

As Figure 4.1(d) and (e) show, the parameter numbers are positively associated with storage usage. But in the same CNNs architecture, increasing model depth could bring significant GFLOPs increase. (see Figure 4.1(c)) Surprisingly, Figure 4.1(a) shows that the same architecture brings almost the same CUDA memory footprint, the compressed models such as MobileNetv2 and ShuffleNetv2 cannot striking decrease the CUDA usage. In this case, ShuffleNetv2 and VGG19_bn consume the lowest and highest resource usage, separately. When using the ARC3 CPU computing node to test the models, the model under the VGG architecture has less difference in memory usage. But the models which introduce the residual theory(He et al., 2016), deeper layer models have a higher memory footprint.

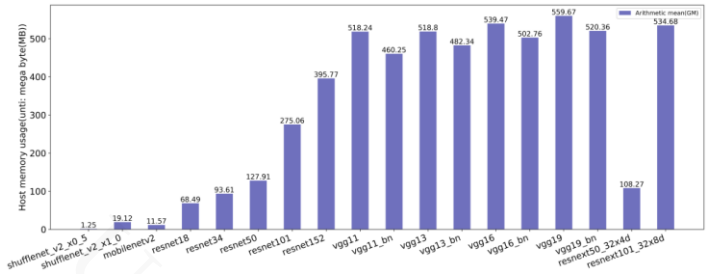
In Figure 4.2(a), when using GPU, along with model layer increase, VGG model inference time is increased but ResNet models inference time is decreased. Compared with uncompressed models, MobileNetv2 and ShuffleNetv2 cannot have very low inference time. ResNet152 achieves the lowest inference time. However, ShuffleNetv2_x0.5 has the lowest resource usage.

Figure 4.2(c) illustrates that when float32 pre-trained models are run by X86 CPU, the inference time of the models which introduce residual learning is related to parameter number. But about VGG architecture models, whether using BN or not significantly affects inference time. About no BN VGG models, the inference time also follows the parameter number. However, VGG13_BN has the highest inference time which peaks at 4.2351 seconds.

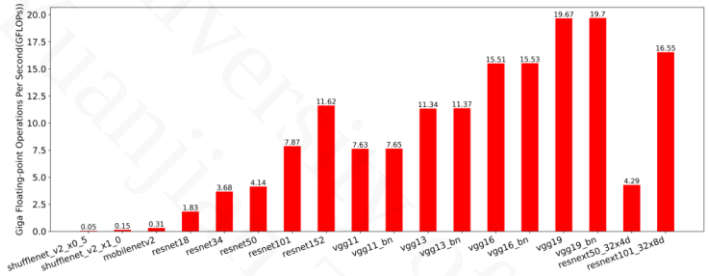
BN significantly affects running time, but not all the cases. As Figure 4.2(d)shows, only Vgg11_BN and VGG13_BN are affected to increase the running time. Especially VGG11_BN, its running time reached at 277.59 seconds. But the highest efficiency model is ShuffleNetv2_0.5 which has a running time of 13.312.



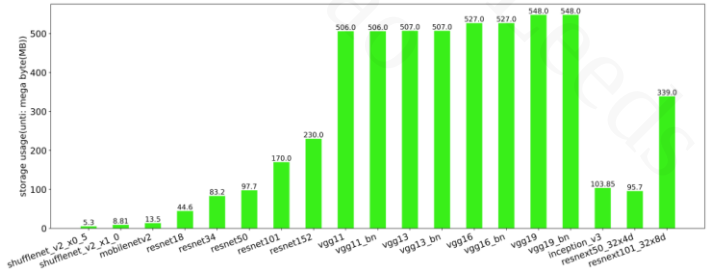
(a) CUDA memory footprint



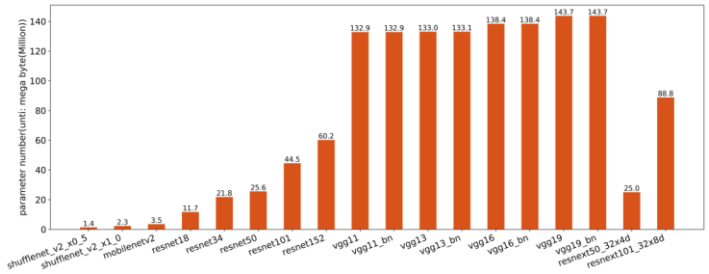
(b) Host memory footprint



(c) GFLOPs

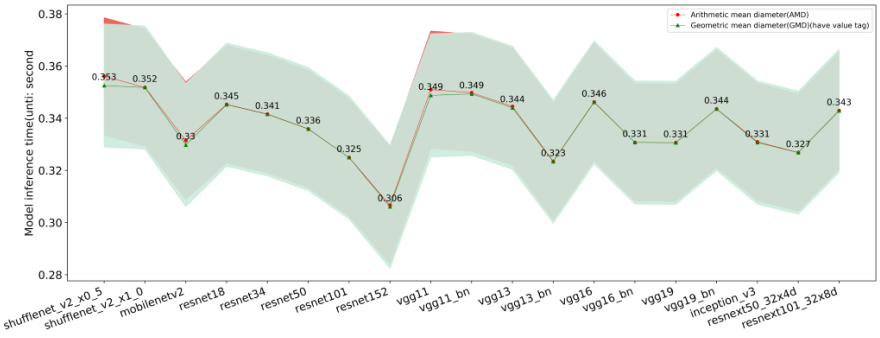


(d) Storage usage

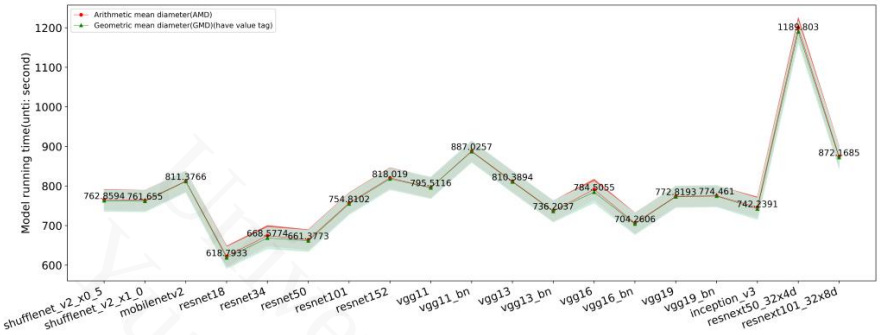


(e) Parameter number

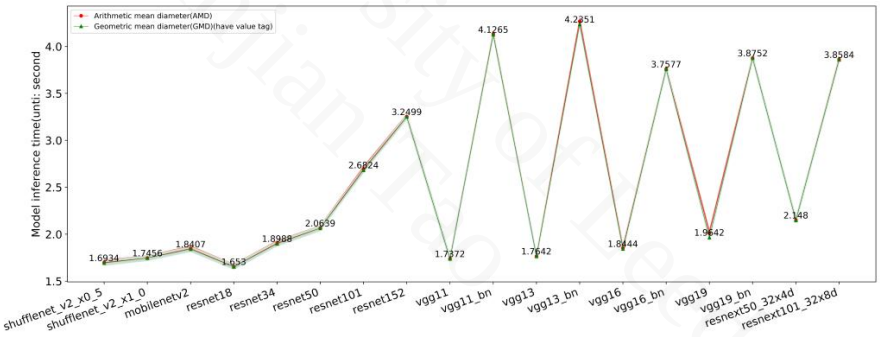
Figure 4.1 The CUDA memory footprint after float32 pre-trained models are loaded (a), the models' memory footprint when they implement by one ARC3 CPU computing node(b), the models estimated GFLOPs (c), practical storage usage (d), and The models' parameter numbers(e)



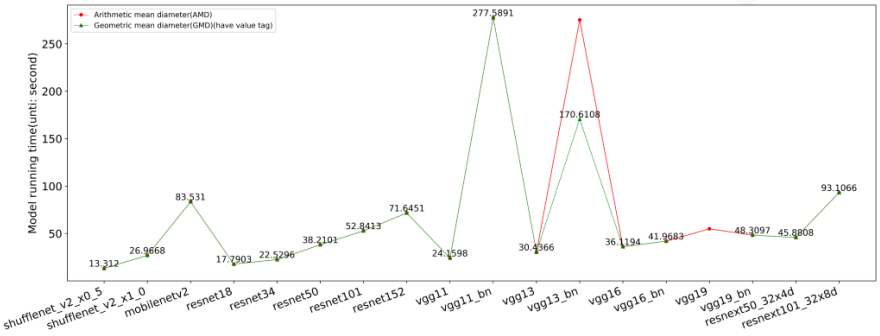
(a) Using CUDA inference time



(b) CUDA running all ImageNet validset time



(c) CPU inference time



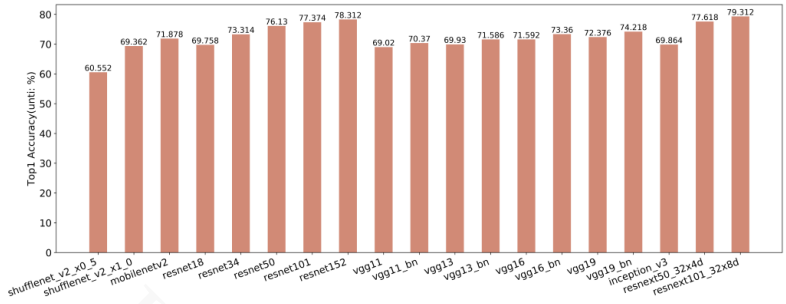
(d) ARC3 CPU Classify 5k full resolution images consumption time

Figure 4.2 V100 GPU test all the ImageNet validset, model inference time(a), and The total time consumption of classifying 5k ImageNet testset full resolution images(b); Using ARC3 CPU node to test the float32 pre-trained models, model inference time(c) and The total time consumption of classifying 5k ImageNet testset full resolution images(d)

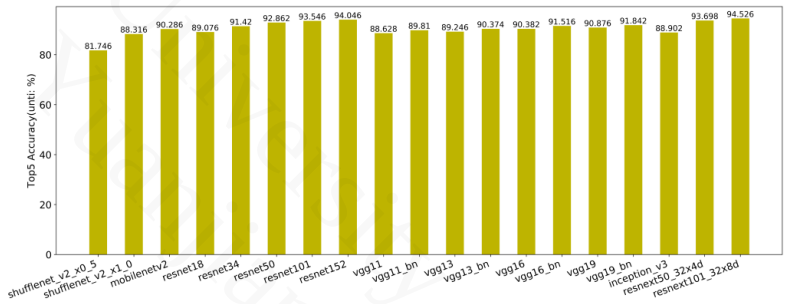
According to Figure 4.3, Benefit from combining state-of-the-art technologies, ResNext101_32x8 owns the highest Top-1 accuracy, Top-5 accuracy, precision, recall, and

F1-score. The worst three models are ShuffleNetv2_0.5, ShuffleNetv2_1.0, and ResNet18, Separately. However, compared with ShuffleNetv2, although MobileNetv2 is a compressed model, its performance is between VGG13_bn and VGG16_bn.

To sum up, V100 can bring significant float32 computing power. In this computing environment, different models also can gain significant efficiency. As a result, If the memory is not very strict, choosing relative complex models like ResNet101 model will get achieve better performance and finish the jobs in a short time.

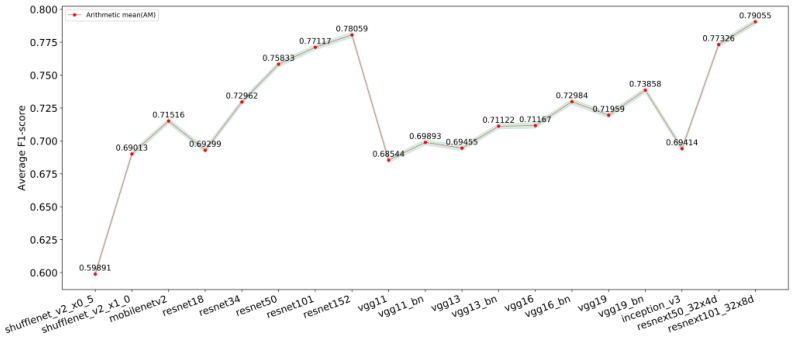


(a) Top1 accuracy



(b) Top5 accuracy

(c) Precision(greenline) and recall(redline)

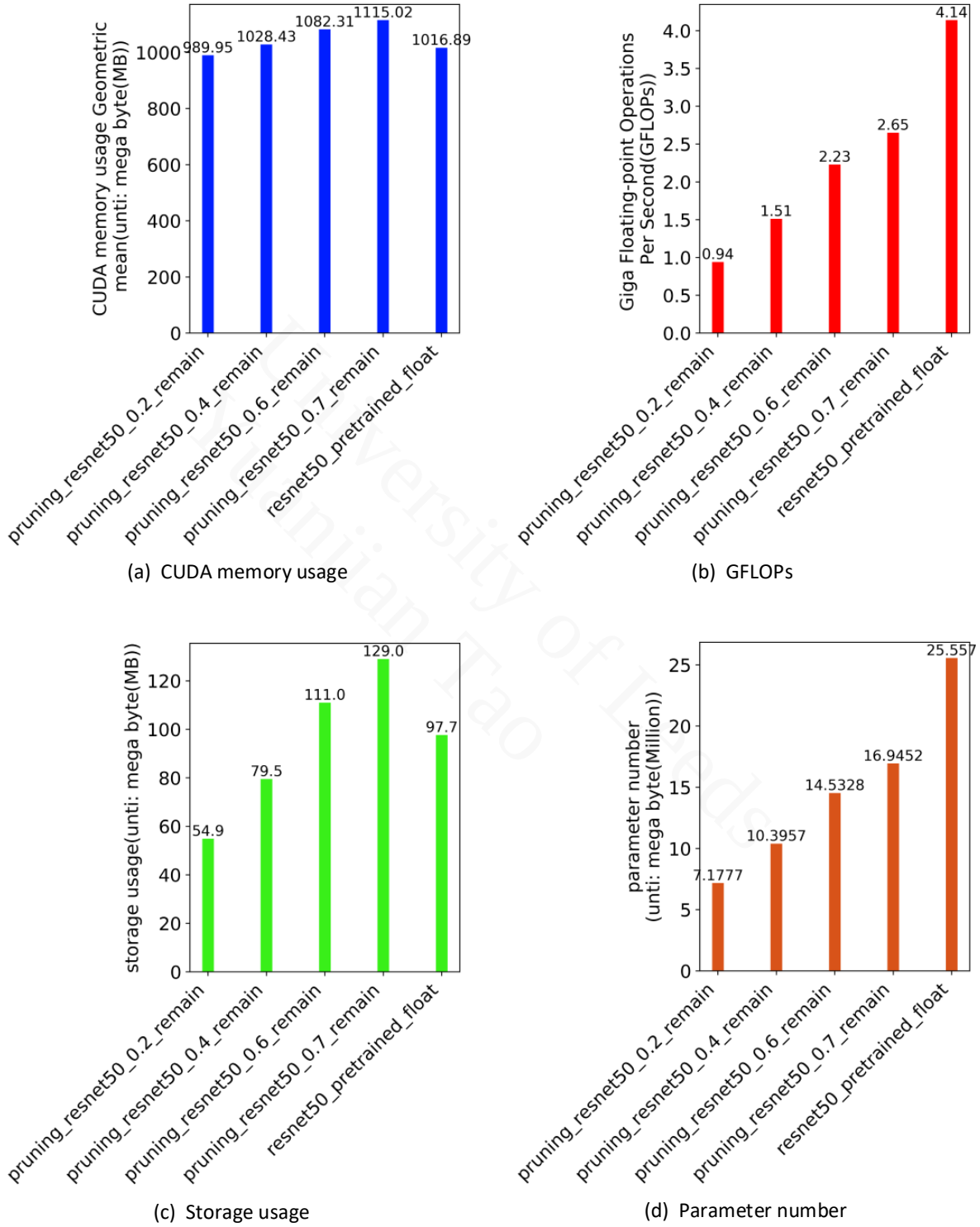


(d) F1-score

Figure 4.3 Each model Top-1 accuracy(a), Top-5 accuracy(b), model precision and recall(c), and F1-score(d)

4.2 Pruned models

In the previous section, ResNet50 get moderate performance and resource usage. After pruning and fine-tune, the different properties are changed. This also affects efficiency and performance.



(a) CUDA memory usage (b) GFLOPs

(c) Storage usage (d) Parameter number

Figure 4.4 The CUDA memory usage after pre-pruned and fine-tuned models are loaded (a), the models estimated GFLOPs (b), practical storage usage (c), and The models' parameter numbers(d)

As Figure 4.4 shows, because pruning introduces weight and filter masks to set some weight results to zero, the CUDA memory and storage usage is growing as the network remains approximately 60% or 70% parameters. However, the parameter number and GFLOPs are declining. Because channels and filters are pruned as well when the pruning parameter rate increases to 80%, Both storage usage, and memory footprint go down.

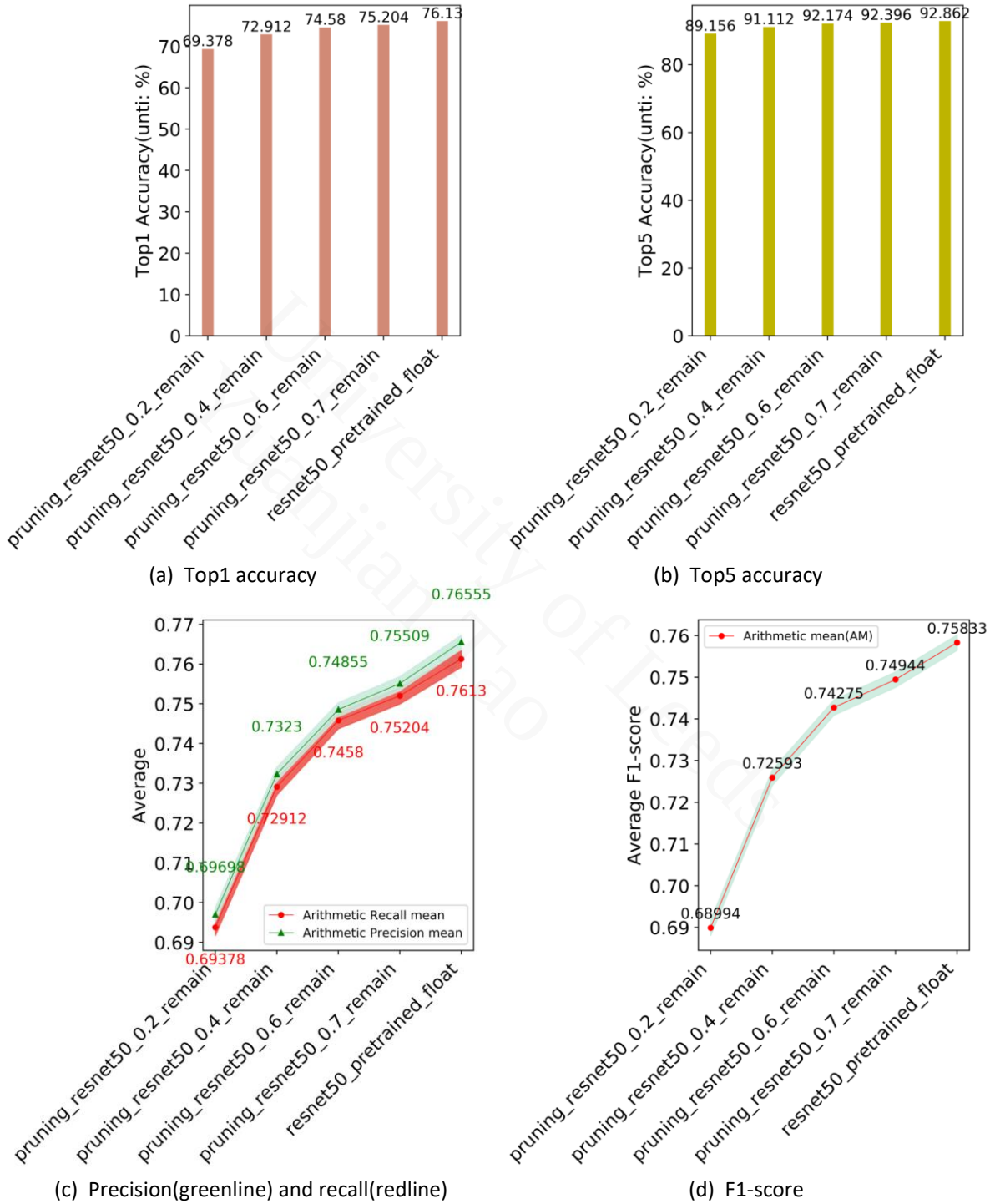
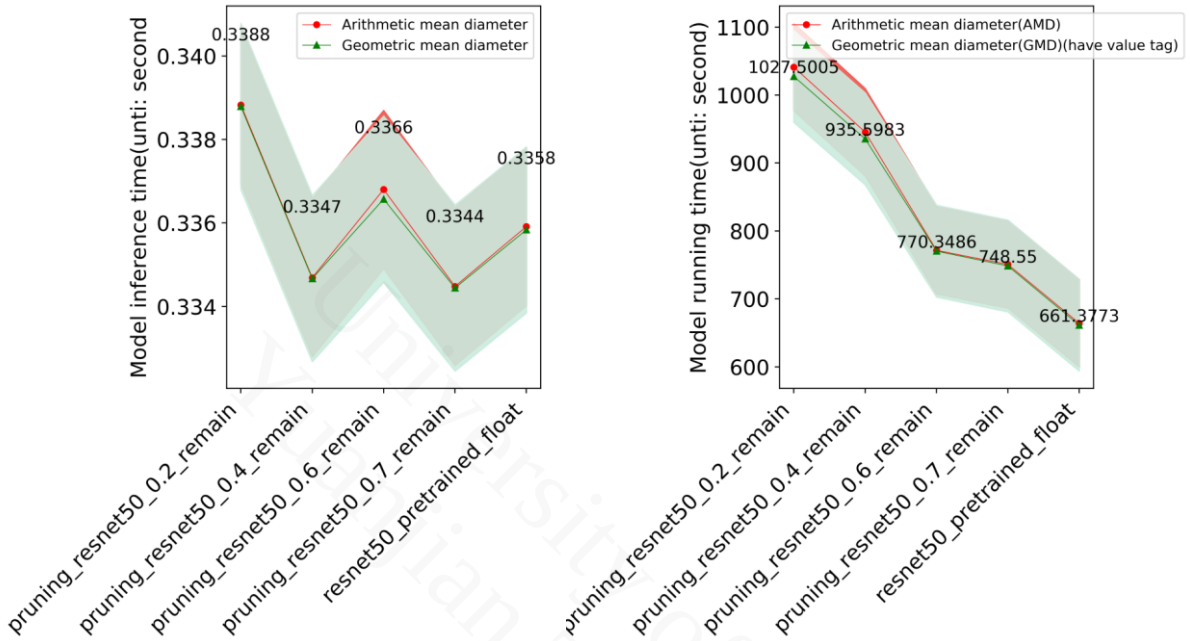


Figure 4.5 Compare different ResNet50 pruning rate and original pre-trained ResNet50 Top-1 accuracy(a), Top-5 accuracy(b), precision, recall(c), and F1-score(d).

Through pruning, the model can use a lower parameter number to achieve almost the same as the original model performance. As Figure 4.5(a) shows, although the accuracy

continually declines after different rate pruning, the Top-1 accuracy loss only is 1.57% from the original model to pruning 40% parameters. Even if only 20% of parameters remain, the accuracy still stays 69.73%. Figure 4.5 (c) and (d) show that average precision, recall, and F1-score decline tendency is similar to Top-1. About Top-5, the accurate decline is smaller than Top-1. For instance, cutting 80% parameters only decrease 3.706% Top-5 accuracy. (see Figure 4.5(b)) As a result, pruning does not decrease model performance significantly even after cutting most of the parameters.



(a) Inference time (b) Classify 5k images consumption time

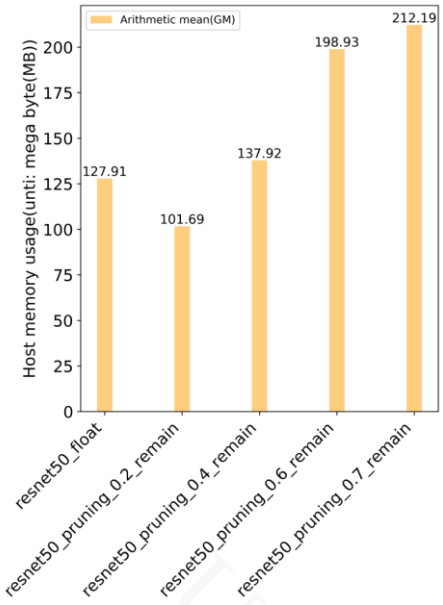
Figure 4.6 Pruning model inference time(a) and classify 5k full resolution images time(b)

As illustrated in Figure 4.6(a), when using CUDA to implement the pruned models, pruning virtually does not change ResNet50 inference time too much. In particular, 20% ResNet50 parameter remaining model has the highest inference time which peaks at 0.3388 seconds. Other test results also within the error range.

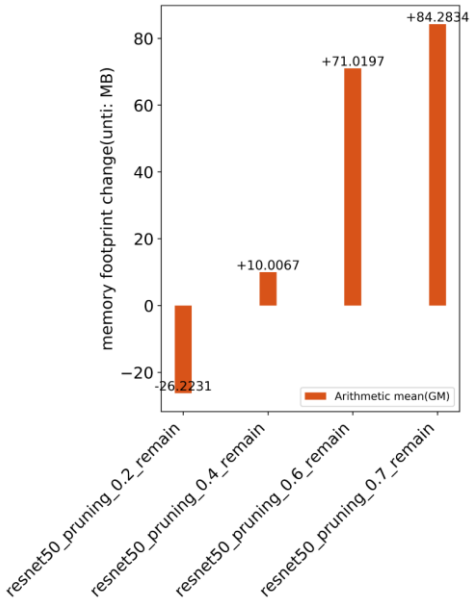
As for Figure 4.6(b), with pruning rate increase, time consumption elevate. because cutting more parameters need more masks to ignore specific parameters. This needs more running time to run these operations.

In order to find out when the computing power is limited, whether the pruning models have better performance or not. One ARC3 23core-128G Intel(R) Xeon(R) CPU E5-2650 v4 node is chosen to be the testbed. Each ResNet50 model is assigned five thousand ImageNet testset images to test how much time the mode needs to run.

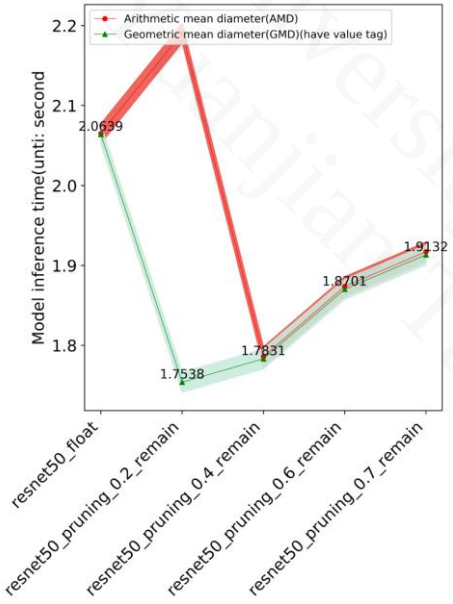
As Figure 4.7 showed, the host memory usage is similar to CUDA.(Figure 4.7(a)) As the pruning rate grows, inference time gradually decreases. But the running time does not have this pattern. As Figure 4.7(f) shows, when ResNet50 remains 60% parameter, it has the highest running efficiency.



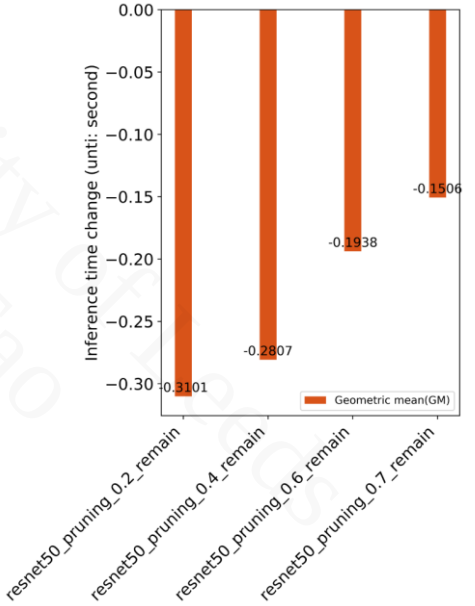
(a) Host memory usage



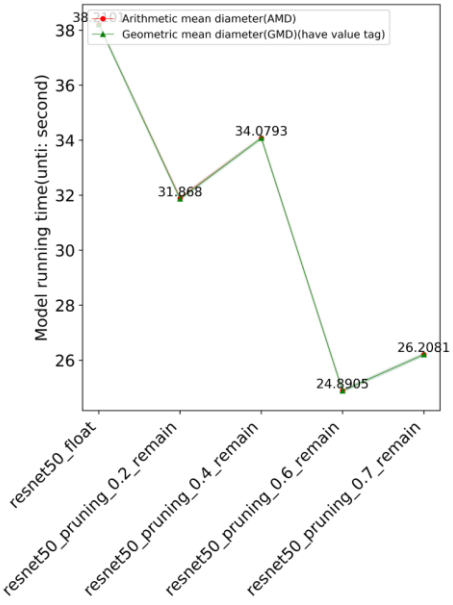
(b) Host memory Change



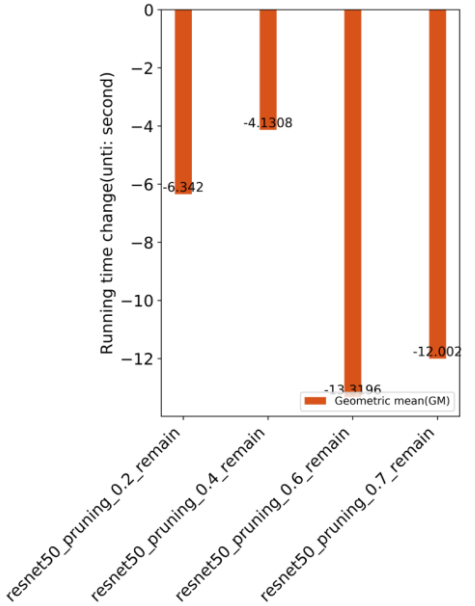
(c) Model inference time



(d) Inference time change



(f) Running time change



(f) Running time change

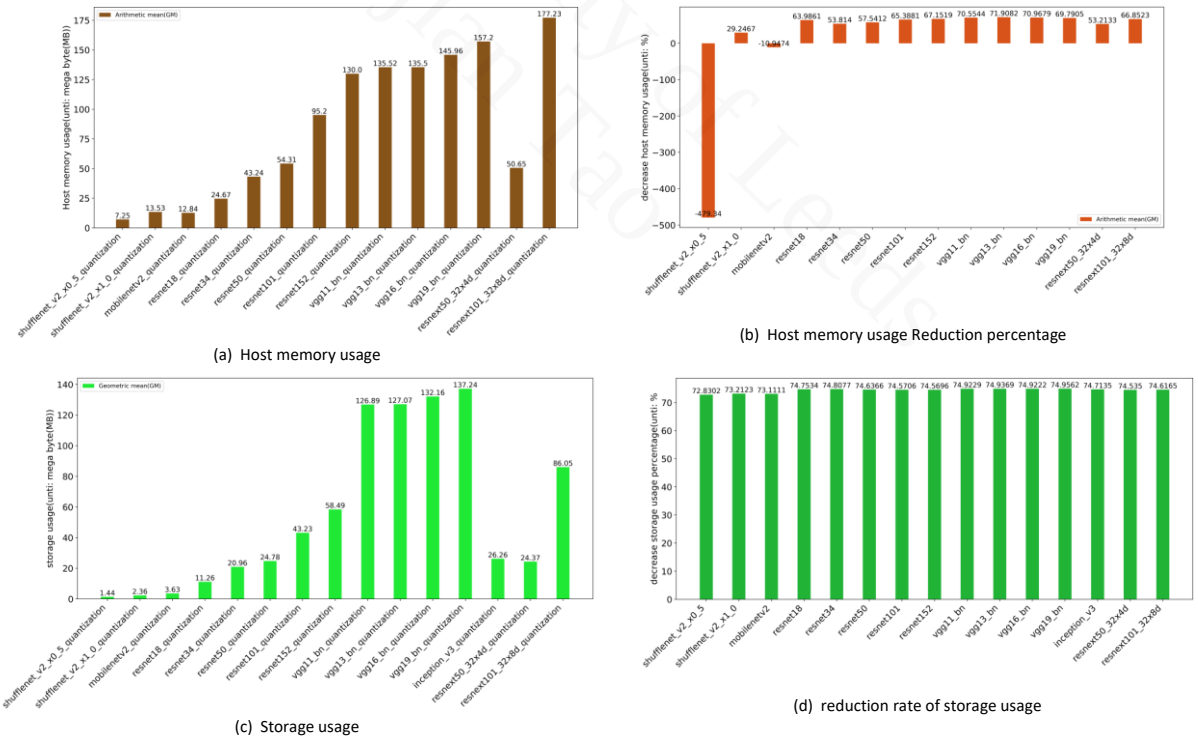
Figure 4.7 Using one ARC3 normal CPU node to test the host memory usage(a), model inference time(c), model running time(e), and their change(b), (d), and (f).

To conclude, pruning cannot decrease memory footprint and storage usage when the pruning rate is not very high. If the running platforms have enough float32 computing power, pruning cannot have a positive effect to reduce inference and running time. But when the computing power of the devices is limited, it not only can significantly reduce the inference and running time but also can keep the model accuracy at a comparable level.

4.3 Quantized models

According to Jacob(Jacob et al., 2018), statistical quantization can transform parameters to lower precision type. In Pytorch, the parameters are transformed from float32 to int8. In theory, the quantized model memory footprint and storage usage only need a quarter of float32 models. In this test, all the host memory usage, inference, and running time are collected by ARC3 CPU testbed. About storage usage and model performance are also collected by ARC3 CPU testbed.

Figure 4.8 shows that except for compressed models such as MobileNetv2 and ShuffleNet, other models can achieve a 75% host memory footprint(see Figure 4.8) and storage usage(see Figure 4.8) reduction close to the theoretical value. Because compressed models have lower basic parameter numbers, the ratio of basic data and parameter storage usage is relatively higher. Compared with VGG and ResNet, this causes the compression rate to be lower.



(a) Host memory usage

(b) Host memory usage Reduction percentage

(c) Storage usage

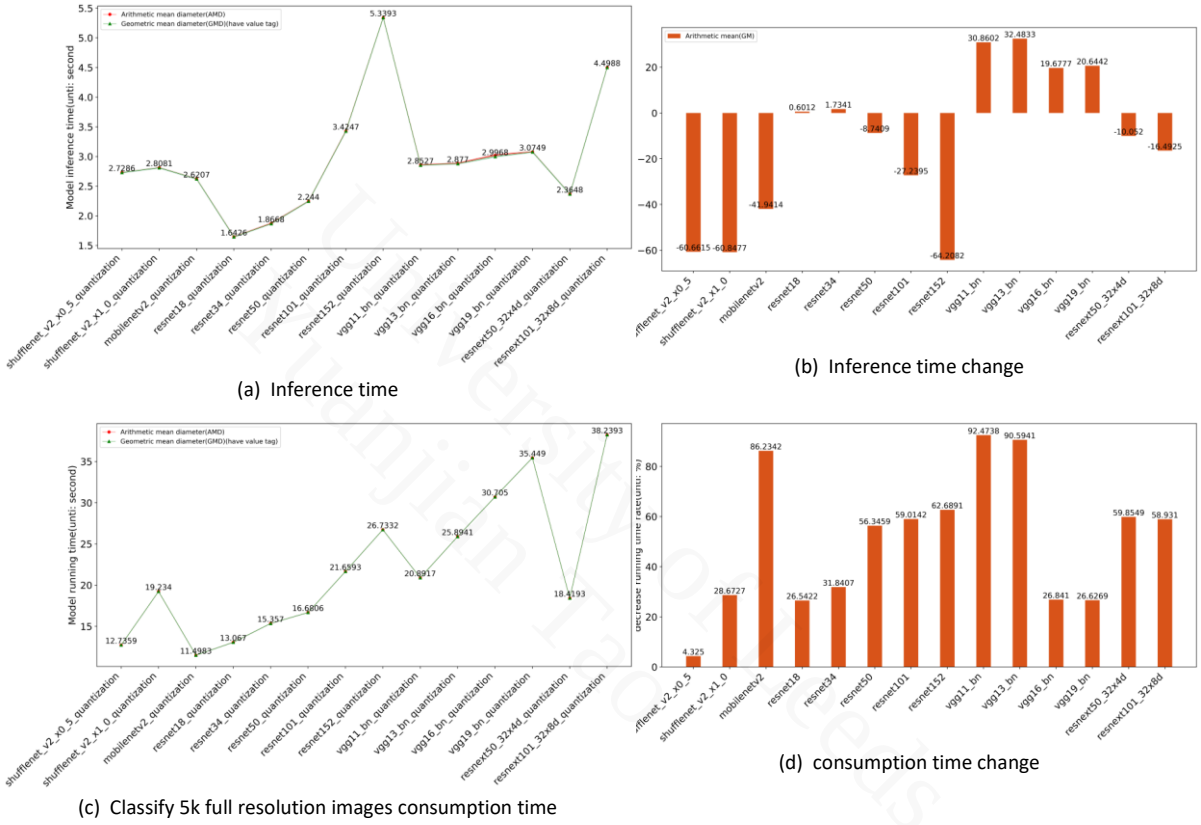
(d) reduction rate of storage usage

Figure 4.8 upper two figures: quantized models host memory footprint(a) and compared with float pre-trained models the reduce percentage of host memory footprint. Lower two figures: quantization models storage space usage(c) and using storage reduce rate(d).

Figure 4.9 shows the inference and running time change. Compressed models and lower layers ResNet display the inference time increase when quantizing the models, in particular, ShuffleNetv2_1.0 has more than doubled the inference time. Other models that introduce

residual theory(He et al., 2016) have varying degrees of inference time increase. (see Figure 4.9(a) and (b)) Especially, deeper residual block models such as ResNext101_32x8d(Mahajan et al., 2018) and ResNet101(He et al., 2016) which have 33 bottlenecks. But all the models gained performance improvements in multiple tasks after quantization. Surprisingly, the MobileNetv2, VGG11_bn, and VGG13_bn have nearly doubled their efficiency.

As a result, quantization cannot cause all the models to increase inference time efficiency, using residual theory and deeper models. But it can significantly increase multiple tasks efficiency.



(a) Inference time

(b) Inference time change

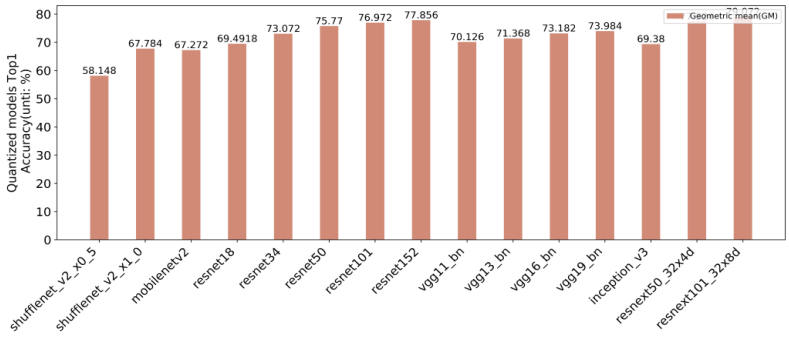
(c) Classify 5k full resolution images consumption time

(d) consumption time change

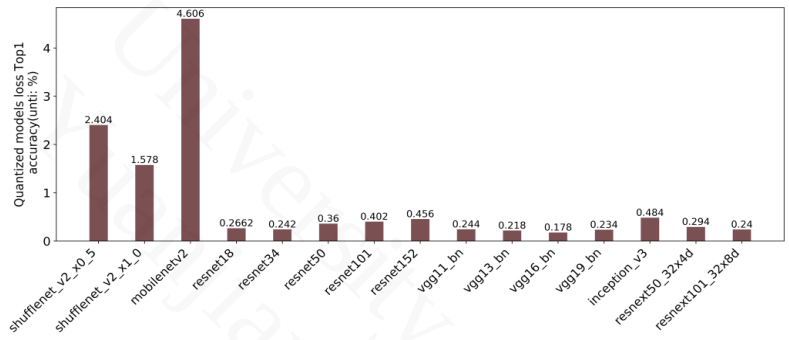
Figure 4.9 upper two figures: quantization models inference time(a) and compared with float32 pre-trained models inference time change(b). lower two figures: each quantization models running five thousand ImageNet testset full resolution image time consumption(c) and compared with float32 pre-trained model running time change(d)

The model performance also could be affected when quantizing the models. As Figure 4.10 shows, all the models have performance loss after quantization. Particularly, compressed models have a significant decrease in all performance indicator. However, traditional models such as VGG and ResNet lose very little performance. The reason is that compressed models have lower parameters. This means that the information of the compressed models already learned is less than higher parameter number models. In order to achieve the same performance as the traditional model, the limited perceptrons and weights needs to learn crucial information. So in compressed models, rate of redundancy and unnecessary perceptrons are very low. When quantizing the compressed models, it means that the rate of crucial information loss is higher than other architecture models. Therefore, compressed models are more affected than other models.

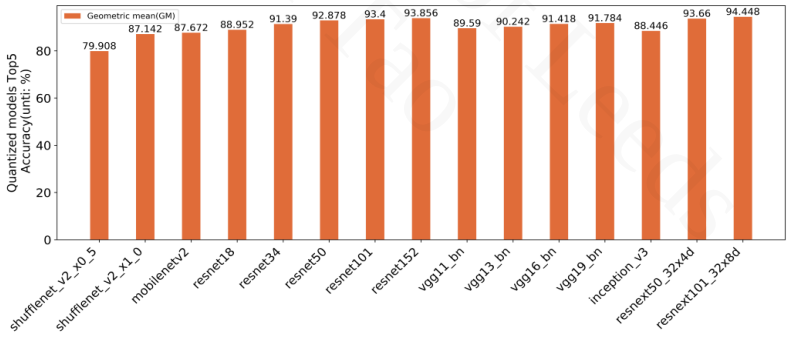
Figure 4.10 and Figure 4.11 also illustrates that MobileNetv2 has the most performance loss including Top-1, Top-5, F1-score, precision, and recall. Its Top-1 accuracy has even dropped below ShuffleNetv2_1.0 and reached 67.272%. Other performance indicators are lower than ShuffleNetv2_1.0. However, Because VGG has a simpler algorithm and structure, it is affected less.



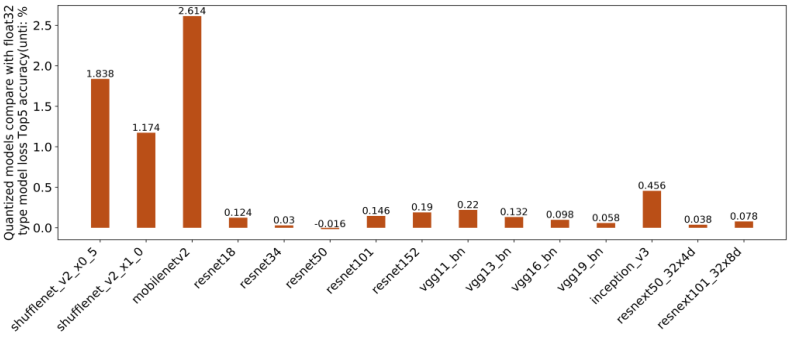
(a) Top1 accuracy



(b) Top1 accuracy reduction



(c) Top5 accuracy



(d) Top5 accuracy reduction

Figure 4.10 Compared with float32, quantization models Top-1 accuracy(a) and change(b); Top-5 accuracy(c) and change(d)

Figure 4.11 Compared with float32, quantized models F1-score(a) and reduce range(b); Recall/precision(c) and recall(d)/precision(e) change.

To conclude, although quantized models cannot be used by normal accelerators like GPU, quantization can significantly reduce memory footprint and model storage size. Also, in batch data processing, quantization can significantly increase the model efficiency. However, it could sacrifice model performance and not every model can gain inference efficiency. As a result, if CNNs are used by ARM or X86 CPU architecture devices and not any float32 accelerators, it may be better choice. But quantized transform may change the performance and characteristics significantly. So when choosing quantized models, the trade-off of performance, hardware, and efficiency needs to be considered.

4.4 Mixed comparison of pruning and quantization

In order to minimize the model hardware demand and find the best efficiency models, in this project, pruning and quantization are compressed ResNet50 to maximize the ResNet50 efficiency. Therefore, different pruning rate models are quantized and evaluate storage usage, time consumption, and performance.

Figure 4.12 shows that the memory footprint and storage usage of the models after the pre-trained ResNet50 is used by quantization or pruning or both. As Figure 4.12(b) and (d) illustrate except ResNet50 which is pruned 30%, 40%, and 60%, other models decrease both the memory footprint and storage usage. Although pruning 60% ResNet50 has some storage decline, the extent is very low. As a result, quantization can ensure models to decrease size and memory footprint. But if pruning intends on doing the same thing, it needs to exceed the pruning rate threshold. If quantization and pruning are used in the same model, the decreasing extent of memory and storage usage is very significant

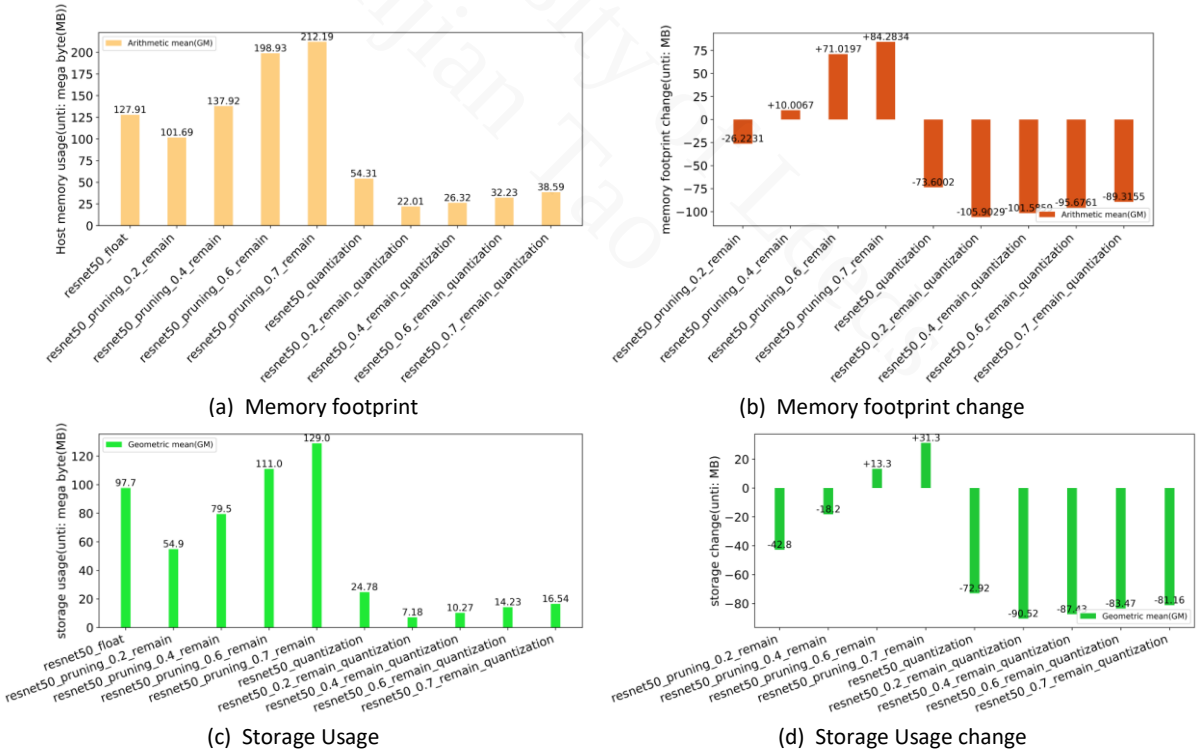
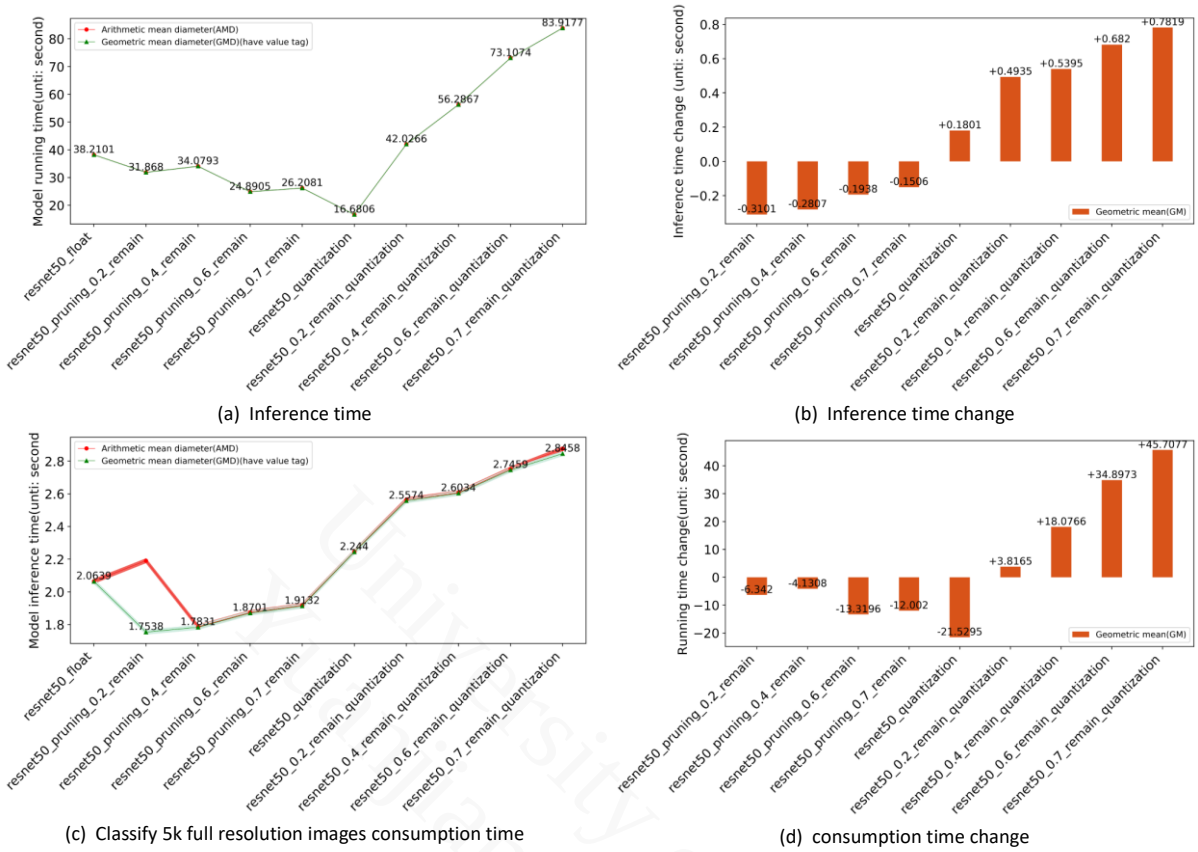


Figure 4.12 upper two figures: pre-trained, pruned and quantized ResNet50 models host memory footprint(a) and compared with float32 pre-trained ResNet50 models the reduced percentage of host memory footprint. Lower two figures: models storage space usage(c) and compared with float32 pre-trained ResNet50 models using storage reduce rate(d).

As Figure 4.13(b) shows that along with the pruning rate increase, the inference time gradually decreases. But the quantization could increase the inference time. As for running

time consumption, the highest efficiency model is unpruned and quantized ResNet50. After quantization, the pruning models instead increase the running time. (see Figure 4.13(d))



(a) Inference time

(b) Inference time change

(c) Classify 5k full resolution images consumption time

(d) consumption time change

Figure 4.13 upper two figures: pre-trained, pruned, and quantized ResNet50 models inference time(a) and compared with float32 pre-trained ResNet50 inference time change(b). lower two figures: each ResNet50 models running five thousand ImageNet testset full resolution image time consumption(c) and compared with float32 pre-trained ResNet50 running time change(d)

Figure 4.14 and Figure 4.15 illustrates that quantization and pruning affect the model's preference. But when the quantization type is int8, the preference affection of quantization is smaller than pruning. Meanwhile, the performance loss of pruned quantized models is more related to pruning than quantization int8.

In short, pruning and quantization have a different characteristic which can optimize different properties of models. If using both of them in the same models, it can significantly reduce the model size and memory footprint. But it could reduce the model efficiency and also sacrifice some extent model performance.

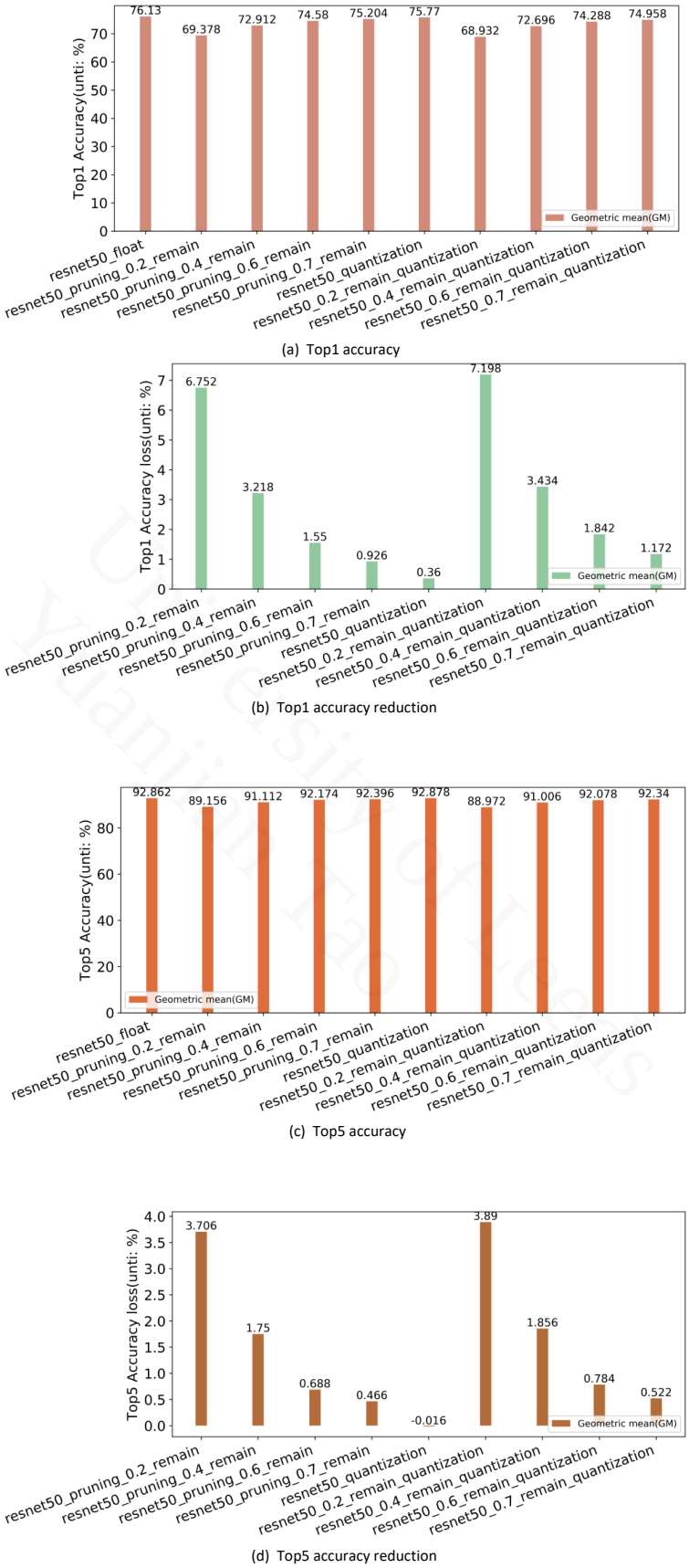
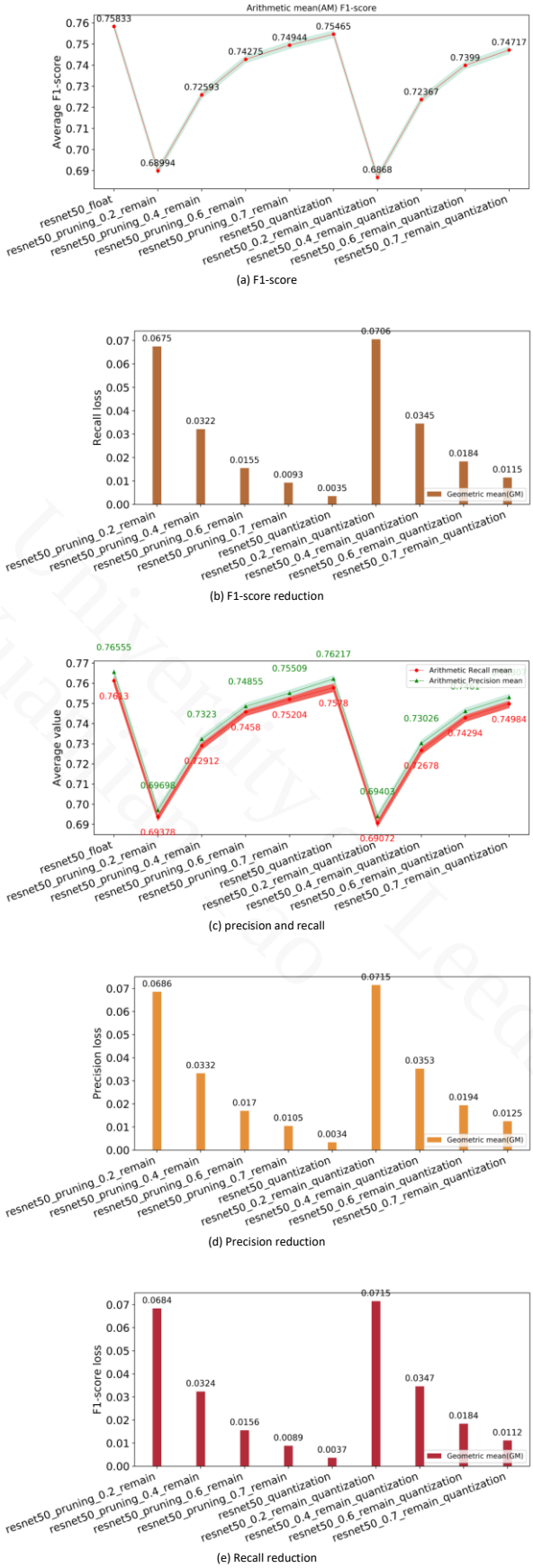


Figure 4.14 models Top-1 accuracy(a), Top-5 accuracy(c), and their reduction rate(b)(d)



(a) F1-score

(b) F1-score reduction

(c) precision and recall

(d) Precision reduction

(e) Recall reduction

Figure 4.15 Compared with ResNet50_float32, F1-score(a), Precision/Recall (c), and their reduction rate(b)(d)(e).

4.5 Horizontal comparison of model performance and efficiency under hardware constraints

In this section, limited hardware is considered to implement the CNNs. Based on all model performance, time consumption, and resource usage(see Appendix A.1 and Appendix A.2) figure out which model has the best specific property and give the recommendation in specific hardware.(see Table 4.1)

Table 4.1 the models with the best performance among different properties

model name	device	best property	best property value	comment
shufflenet_v2_x0_5	cuda	The lowest CUDA memory footprint and running time	CUDAmemory= 924.5328MB running time = 8.639846 seconds	When running model by GPU, it has lowest CUDA memory footprint model, running time, parameter number, and size
resnext101_32x8d	cuda/ cpu	highest Top-1 , Top-5 accuracy, F1-score, recall, and precision	Top-1= 79.312%, Top-5= 94.526%, F1-score=0.7905506, recall=0.79312, precision = 0.7958598	The highest performance CNN model in the experiment
shufflenet_v2_x0_5	cpu	The lowest host memory footprint, GFLOPs, and parameter number	memory footprint = 1.2510937MB, GFLOPs= 0.05, parameter number= 1.3668 million	
shufflenet_v2_x0_5 _ quantization	cpu	The lowest storage usage	1.44MB	using int8 to quantized shufflenet_v2_x0_ 5 can generate the lowest storage usage model
resnext101_32x8d_ quantization	cpu	highest Top-1 , Top-5 accuracy, F1-score, recall, and precision in quantized models	Top-1 = 79.072%, Top-5 = 94.448%, F1-score = 0.7877295, recall = 0.79042, precision = 0.7932688	The highest performance CNN model in the quantized models
resnet18	cuda	Inference time	1.1651825 seconds	

In practice, every property needs to be considered to find appropriate models in application development. Imagine a scenario is that the device does not allocate GPU or other accelerators, meanwhile, it does not have very strict accuracy requirements and its available memory and storage are very limited like Industrial Control Motherboard. My recommendation is MobileNetv2_quantization. It could be the best choice because of the low time and resource usage. In the scenario such as video analysis or intelligent photo classification, the applications are run on a personal computer and also do not allocate GPU or open GPU acceleration, I very recommend quantized resnext50_32x4d. Because compared with ResNet50, it has very close inference and running time but itsTop1 and Top5 accuracy are nearly 2 and 1 percent higher than that of ResNet50. If the personal computer uses CNN to recognize or separate one specific image objective rather than a batch of jobs, meanwhile the hardware cannot be predicted whether allocate the GPU or not, ResNet18

should be the best model in this scenario. Because it not only has the lowest inference time when using CUDA but also when using CPU, it has comparable inference time. Therefore, different application scenarios need to trade-off different properties of models. Also, hardware limitation is a very important factor in model choice.

Chapter 5
Validation of Results

5.1 Testbed

The testbed is one of the most important factors in experiments. Because valid results are based on a stable test environment, it can ensure the results are more valuable. Meanwhile, stable tested can prevent external cases that could affect machine statuses such as unstable electricity and temperature. Also, the experiment needs time, ensuring each experiment item is in the same environment, which is equally important.

In order to ensure the tests are not interfered with. University of Leeds ARC3 and ARC4 are chosen to generate and test all models. They have a professional monitor, power sustainment, and maintenance. Although one node only can be used 48 hours at a time and applying a single core is easier in the multi-user operating system(OS), all the tests only introduce one whole individual node when needed to use CPU. In the test process, ARC4 computing nodes are more stable and efficient to finish the jobs. But it needs to queue a very long time to apply one node. (see Figure 5.1) Therefore, most of the CPU tests are finished by ARC3. But each test is noted the node information in order to ensure every individual test item is tested by the same type of node, dataset, and testsets. (see Figure 5.2) So the comparisons between the models are fair and equal. In practice, more advanced devices and technologies are used more frequently. So in the GPU tests, state-of-the-art V100 GPU is introduced in the float32 models testing. This can make the result more valuable.

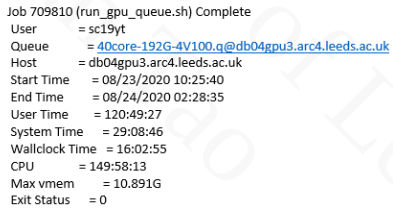


Figure 5.1 ARC4 Sun Grid Engine (SGE) queue

model memory usage	cross_enti	model cud	Testbed cpu information
-2.703125			cpu counts: 24, cpu model: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
28.80078125			cpu counts: 24, cpu model: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
34.81640625			cpu counts: 24, cpu model: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
34.05859375			cpu counts: 24, cpu model: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz
-10.07421875			cpu counts: 24, cpu model: Intel(R) Xeon(R) CPU E5-2650 v4 @ 2.20GHz

Figure 5.2 Log testbed information in each test.

5.2 Data collection and analysis

In experiments, unpredictable interruption and data fluctuation are inevitable. So repeating experiments and scientific summarization can minimize their effect.

Regarding dataset selection, because all the models are pre-trained by ILSVRC2012 dataset which includes one thousand classifications. If the testset cannot cover all the ImageNet 1000 classes or each class only has one or two images to be tested. The results of model performance are not accurate. Therefore, in this project, all the ILSVRC2012 validset is used to test model performance even though it needs more time in every test epoch. This produces test results that are valuable and closer to another more professional test.

About data analysis, eliminating too high or low recorded values is one of the very important methods to avoid perturbation. Therefore, every individual experiment is run dozens of times recursively. Because of available hardware limitations, test model performance to collect such Top-1 accuracy, Top-5 accuracy, precision, recall, and F1-score is run and collected 20 times. But comparing model efficiency between models which only need to finish the same jobs in the same environment, an experiment is run 100 times. All the recursive collection data is averaged by geometric or arithmetic mean and given the 90% confidence interval. (Seixas et al., 1988) This can reduce affection from data perturbation.

Python was selected as a tool for data analysis. Because compared with manual calculation or automatic tools like Excel, Python can trace the data flow, generating algorithms, calculational logic, and plot figures logic easier. Also, when the experiment data or analysis method are changed, all the analysis parameters and algorithms are modified easier. This can significantly reduce the data analysis works and increase the efficiency in reviewing all the analysis processes and ensuring the data is correct.

5.3 Compare results with other same researches

This section focuses on comparing the results corrected by this project and other same researches to demonstrate that the results are valid.

As Table 5.1 shows that the comparison of performance evaluation of the pre-trained model between this project and Pytorch Model Zoo(Facebook), the results are very close between two different research experiments. The only major difference is the exact number after the decimal point.

Table 5.1 Compared the model performance results between this project and Pytorch Model zoo(**Facebook**).

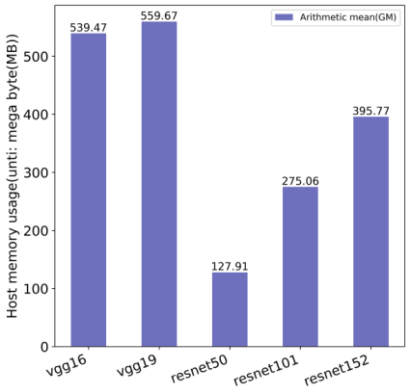
model name	Top-1 accuracy(this project unit: %)	Top-1 accuracy(Pytorch Hub unit: %)	Top-5 accuracy(this project unit: %)	Top-5 accuracy(Pytorch Hub unit: %)
shufflenet_v2_x1_0	69.362	69.36	88.316	88.32
mobilenetv2	71.878	71.88	90.286	90.29
resnet18	69.758	69.76	89.076	89.08
resnet34	73.314	73.3	91.42	91.42
resnet50	76.13	76.15	92.862	92.87
resnet101	77.374	77.37	93.546	93.56
resnet152	78.312	78.31	94.046	94.06
vgg11	69.02	69.02	88.628	88.63
vgg11_bn	70.37	73.3	89.81	91.42
vgg13	69.93	69.93	89.246	89.25
vgg13_bn	71.586	71.55	90.374	90.37
vgg16	71.592	71.59	90.382	90.38
vgg16_bn	73.36	73.37	91.516	91.5
vgg19	72.376	72.38	90.876	90.88
vgg19_bn	74.218	74.24	91.842	91.85
resnext50_32x4d	77.618	77.62	93.698	93.7
resnext101_32x8d	79.312	79.31	94.526	94.53

Table 5.2 shows that the parameter number, GFLOPs, Top-1, and Top-5 accuracy between tested by this project and the research of Lin et al. (Lin et al., 2020) Although the accuracy evaluation has certain deviations, these deviations are within the error ranges. The reason is that the test dataset may be different.

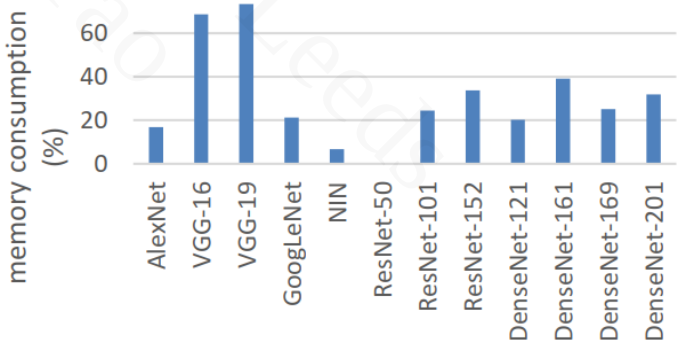
Table 5.2 Compared the model performance results between this project and research from Li et al. (**Lin et al., 2020**)

model name	parameter number (this project unit: million)	parameter number(Li et al. unit: million)	GFLOPs(this project)	GFLOPs(Li et al.)	Top-1 accuracy(this project %)	Top-1 accuracy(Li et al. unit:%)	Top-5 accuracy(this project %)	Top-5 accuracy(Li et al. unit:%)
pruning_resnet50_0.2_remain	7.1777	7.18	0.94	0.93	69.378	69.43	89.156	89.23
pruning_resnet50_0.4_remain	10.3957	10.4	1.51	1.51	72.912	73.04	91.112	91.18
pruning_resnet50_0.6_remain	14.5328	14.53	2.23	2.23	74.58	74.68	92.174	92.17
pruning_resnet50_0.7_remain	16.9452	16.95	2.65	2.64	75.204	75.22	92.396	92.41

Figure 5.3(b) illustrates the memory consumption of VGG16 in the research of Muhammed et al. (Muhammed et al., 2017) is 3.03% less than VGG19. These results are almost same as the Figure 5.3(a) which is 3.6%. Also, the ResNet101 is less 27.6% than ResNet152 in (b). This result also is close to 30.5% in (a).



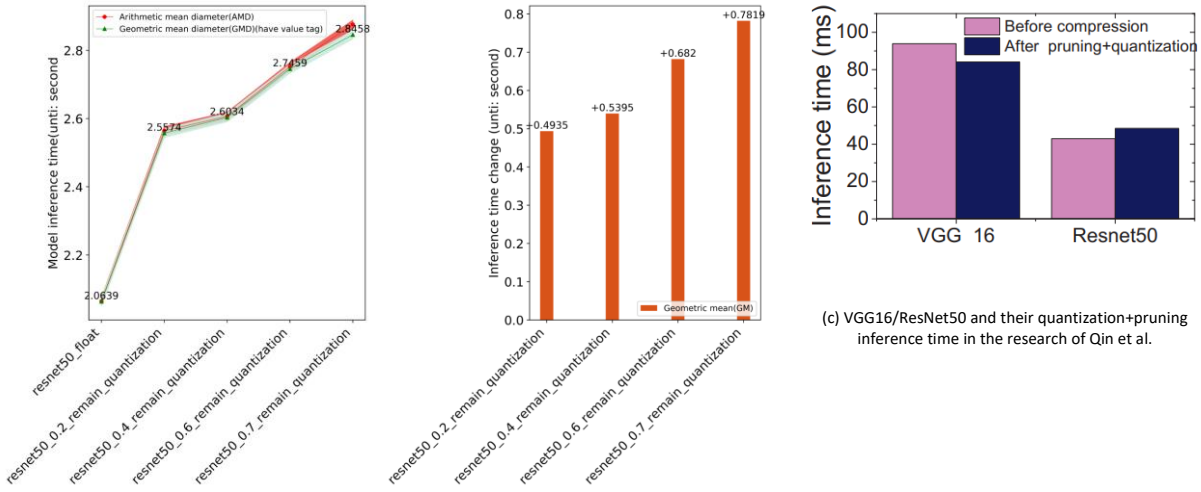
(a) Memory footprint statistic in this project



(b) Memory consumption is log by Muhammed et al.

Figure 5.3 Compared the memory footprint statistic between this project(a) and Muhammed et al. (**Muhammed et al., 2017**)

In Figure 5.4(b), Qin et al. finds that after pruning and quantization, the inference time of ResNet50 increase to approximate 20%. Its result is very close to the result of quantized ResNet50_0.2_remain in this project which increases by 24%.



(a) ResNet50 and quantized pruned ResNet50 inference time in this project

(b) ResNet50 and quantized pruned ResNet50 inference time change in this project

(c) VGG16/ResNet50 and their quantization+pruning inference time in the research of Qin et al.

Figure 5.4 ResNet50 and quantized pruned ResNet50 inference time(a), inference time change(b), and quantization + pruning VGG16/ResNet50 inference time change(c) from the research of Qin et al. (**Qin et al., 2018**)

In conclusion, different statistic result samples are very close to previous research results. So the results which are collected in this project should be valid.

5.4 Conclusion

In order to maximally increase the accuracy of data. I utilize ARC testbed and test the models dozens of time to collect raw testing results. In analysis, setting confidence interval and using different average algorithm also can avoid the unusual data to disturb the final results. Finally, after testing, comparing the results between this project and other researches also can help me to fix the errors in test or analysis.

Chapter 6
Conclusions and Future Work

6.1 Conclusions

These projects evaluate different CNN architectures which base off the original design or are the process by quantization and pruning. The conclusions of performance and efficiency for CNN and compressed technology are representation are as follows:

i. Under the same model architecture, a shallower network depth can be designed to achieve a balance between model efficiency and performance.
ii. Pruning can significantly decrease inference and running time. But when the pruning rate is not high enough, the memory footprint and storage usage could increase.
iii. Quantization can significantly reduce the model size, memory footprint, and running time. But only the models which introduce residual theory(He et al., 2016) can keep or reduce the inference time when the quantized type is int8.
iv. Mixing different compression technologies such as quantizing compressed models or quantizing pruned models may not bring the overall model energy efficiency gains.
v. A model that pursues the ultimate performance of a certain attribute usually sacrifices other attributes significantly. Therefore, when choosing a model, it is necessary to weigh every Key Performance Parameter.

However, there are some flaws and deficiencies in this project. To begin with, because of the project time limit and ARC queue waiting time, the models cannot be tested by different GPUs and CPUs. Moreover, horizontal tests of quantized models should use ARM devices which are used quantized models more generally in practice. Meanwhile, the testbed is based on KVM virtual machine. It may have a certain performance reduction and be affected by other processes from other users. So these tests should be based on independent equipment. This can minimize external interference. Plus, because of hardware and technical limitations, pruning cannot generate and test different model architecture. Meanwhile, quantization only can transform the parameter type from float32 to int8. Also, the dynamic quantization and quantization fine-tune cannot be implemented and tested in their models. Finally, some hypothesize such residual theory could affect the inference time of quantized models that have not been proved.

6.2 Future Work

This project has some flaws and content as the previous section discusses which can be fixed and added it later. In the beginning, various individual hardware and limited computing power devices should include testbeds in order to test the efficiency of the models in various hardware environments. This includes using different technologies to compress different architecture models and testing them. Finally, the log collection should be more detailed. For example, the time recording should be accurate for each layer or operation. This can help to prove the hypothesis of research and give detailed guidance when using compression technologies.

List of References

ABADI, M., AGARWAL, A., BARHAM, P., BREVDO, E., CHEN, Z., CITRO, C., CORRADO, G. S., DAVIS, A., DEAN, J. & DEVIN, M. 2016. Tensorflow: Large-scale machine learning on heterogeneous distributed systems. *arXiv preprint arXiv:1603.04467*.

ABADI, M., ISARD, M. & MURRAY, D. G. A computational model for TensorFlow: an introduction. Proceedings of the 1st ACM SIGPLAN International Workshop on Machine Learning and Programming Languages, 2017. 1-7.

BULAT, A. *pytorch-estimate-flops* [Online]. Available: https://github.com/1adrianb/pytorch-estimate-flops [Accessed].

CENTOS. 2019. *CentOS-7 (1810) Release Notes* [Online]. Available: https://wiki.centos.org/Manuals/ReleaseNotes/CentOS7.1810 [Accessed].

CHANDEL, S. *Keras style model.summary() in PyTorch* [Online]. Available: https://github.com/sksq96/pytorch-summary [Accessed].

CONTRIBUTORS, P. PyTorch online documentation.

DENIL, M., SHAKIBI, B., DINH, L., RANZATO, M. A. & DE FREITAS, N. Predicting parameters in deep learning. Advances in neural information processing systems, 2013. 2148-2156.

DENTON, E. L., ZAREMBA, W., BRUNA, J., LECUN, Y. & FERGUS, R. Exploiting linear structure within convolutional networks for efficient evaluation. Advances in neural information processing systems, 2014. 1269-1277.

DIXON, M. 2016. *ARC3* [Online]. Available: https://arc.leeds.ac.uk/wp-content/uploads/2016/07/ARC3-Service.pdf [Accessed].

DUNDAR, A., JIN, J., GOKHALE, V., KRISHNAMURTHY, B., CANZIANI, A., MARTINI, B. & CULURCIELLO, E. Accelerating deep neural networks on mobile processor with embedded programmable logic. Neural information processing systems conference (NIPS), 2013.

FACEBOOK. *PYTORCH HUB* [Online]. Available: https://pytorch.org/hub/ [Accessed].

FRANKLE, J. & CARBIN, M. 2018. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*.

GONG, Y., LIU, L., YANG, M. & BOURDEV, L. 2014. Compressing deep convolutional networks using vector quantization. *arXiv preprint arXiv:1412.6115*.

HAEFFELE, B. D. & VIDAL, R. Global optimality in neural network training. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017. 7331-7339.

HE, K., ZHANG, X., REN, S. & SUN, J. Deep residual learning for image recognition. Proceedings of the IEEE conference on computer vision and pattern recognition, 2016. 770-778.

HOWARD, A. G., ZHU, M., CHEN, B., KALENICHENKO, D., WANG, W., WEYAND, T., ANDREETTO, M. & ADAM, H. 2017. Mobilenets: Efficient convolutional neural networks for mobile vision applications. *arXiv preprint arXiv:1704.04861*.

HU, H., PENG, R., TAI, Y.-W. & TANG, C.-K. 2016. Network trimming: A data-driven neuron pruning approach towards efficient deep architectures. *arXiv preprint arXiv:1607.03250*.

HUBEL, D. H. & WIESEL, T. N. 1968. Receptive fields and functional architecture of monkey striate cortex. *The Journal of physiology,* 195**,** 215-243.

JACOB, B., KLIGYS, S., CHEN, B., ZHU, M., TANG, M., HOWARD, A., ADAM, H. & KALENICHENKO, D. Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2018. 2704-2713.

JAIN, A., BHATTACHARYA, S., MASUDA, M., SHARMA, V. & WANG, Y. 2020. Efficient Execution of Quantized Deep Learning Models: A Compiler Approach. *arXiv preprint arXiv:2006.10226*.

KRIZHEVSKY, A. 2014. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*.

KRIZHEVSKY, A. & HINTON, G. 2009. Learning multiple layers of features from tiny images.
LECUN, Y., BOTTOU, L., BENGIO, Y. & HAFFNER, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE,* 86**,** 2278-2324.
LI, H., KADAV, A., DURDANOVIC, I., SAMET, H. & GRAF, H. P. 2016. Pruning filters for efficient convnets. *arXiv preprint arXiv:1608.08710*.
LIN, M., JI, R., LI, S., YE, Q., TIAN, Y., LIU, J. & TIAN, Q. 2020. Filter sketch for network pruning. *arXiv preprint arXiv:2001.08514*.
LIU, Z., LI, J., SHEN, Z., HUANG, G., YAN, S. & ZHANG, C. Learning efficient convolutional networks through network slimming. Proceedings of the IEEE International Conference on Computer Vision, 2017. 2736-2744.
MA, N., ZHANG, X., ZHENG, H.-T. & SUN, J. Shufflenet v2: Practical guidelines for efficient cnn architecture design. Proceedings of the European conference on computer vision (ECCV), 2018. 116-131.
MAHAJAN, D., GIRSHICK, R., RAMANATHAN, V., HE, K., PALURI, M., LI, Y., BHARAMBE, A. & VAN DER MAATEN, L. Exploring the limits of weakly supervised pretraining. Proceedings of the European Conference on Computer Vision (ECCV), 2018. 181-196.
MUHAMMED, M. A. E., AHMED, A. A. & KHALID, T. A. Benchmark analysis of popular imagenet classification deep cnn architectures. 2017 International Conference On Smart Technologies For Smart Nation (SmartTechCon), 2017. IEEE, 902-907.
NARASIMHAN, S. Nvidia clocks world's fastest bert training time and largest transformer based model, paving path for advanced conversational ai. Technical report, 2019. URL https://devblogs. nvidia. com/training-bert-with ….
PASZKE, A., GROSS, S., CHINTALA, S., CHANAN, G., YANG, E., DEVITO, Z., LIN, Z., DESMAISON, A., ANTIGA, L. & LERER, A. 2017. Automatic differentiation in pytorch.
PASZKE, A., GROSS, S., MASSA, F., LERER, A., BRADBURY, J., CHANAN, G., KILLEEN, T., LIN, Z., GIMELSHEIN, N. & ANTIGA, L. Pytorch: An imperative style, high-performance deep learning library. Advances in neural information processing systems, 2019. 8026-8037.
QIN, Q., REN, J., YU, J., WANG, H., GAO, L., ZHENG, J., FENG, Y., FANG, J. & WANG, Z. To compress, or not to compress: Characterizing deep learning model compression for embedded inference. 2018 IEEE Intl Conf on Parallel & Distributed Processing with Applications, Ubiquitous Computing & Communications, Big Data & Cloud Computing, Social Computing & Networking, Sustainable Computing & Communications (ISPA/IUCC/BDCloud/SocialCom/SustainCom), 2018. IEEE, 729-736.
RAGHURAMAN KRISHNAMOORTHI, S. W. *STATIC QUANTIZATION WITH EAGER MODE IN PYTORCH* [Online]. Available: https://pytorch.org/tutorials/advanced/static_quantization_tutorial.html#beta-static-quantization-with-eager-mode-in-pytorch [Accessed].
REUTHER, A., MICHALEAS, P., JONES, M., GADEPALLY, V., SAMSI, S. & KEPNER, J. 2019. Survey and benchmarking of machine learning accelerators. *arXiv preprint arXiv:1908.11348*.
RUSSAKOVSKY, O., DENG, J., SU, H., KRAUSE, J., SATHEESH, S., MA, S., HUANG, Z., KARPATHY, A., KHOSLA, A. & BERNSTEIN, M. 2015. Imagenet large scale visual recognition challenge. *International journal of computer vision,* 115**,** 211-252.
SANDLER, M., HOWARD, A., ZHU, M., ZHMOGINOV, A. & CHEN, L.-C. Mobilenetv2: Inverted residuals and linear bottlenecks. Proceedings of the IEEE conference on computer vision and pattern recognition, 2018. 4510-4520.
SANH, V., DEBUT, L., CHAUMOND, J. & WOLF, T. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.
SASAKI, Y. 2007. The truth of the f-measure. 2007.

SEIXAS, N. S., ROBINS, T. G. & MOULTON, L. H. 1988. The use of geometric and arithmetic mean exposures in occupational epidemiology. *American journal of industrial medicine,* 14**,** 465-477.

SIMONYAN, K. & ZISSERMAN, A. 2014. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*.

SRINIVAS, S. & BABU, R. V. 2015. Data-free parameter pruning for deep neural networks. *arXiv preprint arXiv:1507.06149*.

SZEGEDY, C., LIU, W., JIA, Y., SERMANET, P., REED, S., ANGUELOV, D., ERHAN, D., VANHOUCKE, V. & RABINOVICH, A. Going deeper with convolutions. Proceedings of the IEEE conference on computer vision and pattern recognition, 2015. 1-9.

TöLKE, J. 2010. Implementation of a Lattice Boltzmann kernel using the Compute Unified Device Architecture developed by nVIDIA. *Computing and Visualization in Science,* 13**,** 29.

XIE, S., GIRSHICK, R., DOLLáR, P., TU, Z. & HE, K. Aggregated residual transformations for deep neural networks. Proceedings of the IEEE conference on computer vision and pattern recognition, 2017. 1492-1500.

YANG, Z., SHOU, L., GONG, M., LIN, W. & JIANG, D. Model compression with two-stage multi-teacher knowledge distillation for web question answering system. Proceedings of the 13th International Conference on Web Search and Data Mining, 2020. 690-698.

ZHANG, X., ZHOU, X., LIN, M. & SUN, J. Shufflenet: An extremely efficient convolutional neural network for mobile devices. Proceedings of the IEEE conference on computer vision and pattern recognition, 2018. 6848-6856.

Appendix A
External Materials

A.1 All model performance

model name	device type	parameter number (unit:million)	GFLOPs	Top-1 accuracy(%)	Top-5 accuracy(%)	F1-score	Recall	Precision
mobilenetv2	cpu	3.5049	0.31	71.878	90.286	0.7151568	0.71878	0.7213048
mobilenetv2_quantization	cpu	3.5049	0.31	67.272	87.672	0.6696455	0.67312	0.6806347
pruning_resnet50_0.2_remain	cpu	7.1777	0.94	69.378	89.156	0.689938	0.69378	0.6969834
pruning_resnet50_0.4_remain	cpu	10.3957	1.51	72.912	91.112	0.7259277	0.72912	0.7322989
pruning_resnet50_0.6_remain	cpu	14.5328	2.23	74.58	92.174	0.7427464	0.7458	0.7485525
pruning_resnet50_0.7_remain	cpu	16.9452	2.65	75.204	92.396	0.7494368	0.75204	0.7550905
resnet101	cpu	44.5492	7.87	77.374	93.546	0.7711684	0.77374	0.7778877
resnet101_quantization	cpu	44.5492	7.87	76.972	93.4	0.7667998	0.7695	0.7738709
resnet152	cpu	60.1928	11.62	78.312	94.046	0.7805866	0.78312	0.7869845
resnet152_quantization	cpu	60.1928	11.62	77.856	93.856	0.7762495	0.77906	0.7832375
resnet18	cpu	11.6895	1.83	69.758	89.076	0.6929891	0.69758	0.7009789
resnet18_quantization	cpu	11.6895	1.83	69.4918	88.952	0.6908906	0.69534	0.6989566
resnet34	cpu	21.7977	3.68	73.314	91.42	0.7296244	0.73314	0.7368713
resnet34_quantization	cpu	21.7977	3.68	73.072	91.39	0.7270876	0.73056	0.7344773
resnet50	cpu	25.557	4.14	76.13	92.862	0.7583254	0.7613	0.7655472
resnet50_0.2_remain_quantization	cpu	7.1777	0.94	68.932	88.972	0.6867981	0.69072	0.6940338
resnet50_0.4_remain_quantization	cpu	10.3957	1.51	72.696	91.006	0.7236741	0.72678	0.7302559
resnet50_0.6_remain_quantization	cpu	14.5328	2.23	74.288	92.078	0.7398953	0.74294	0.7460997
resnet50_0.7_remain_quantization	cpu	16.9452	2.65	74.958	92.34	0.7471714	0.74984	0.7530696
resnet50_quantization	cpu	25.557	4.14	75.77	92.878	0.7546544	0.7578	0.7621657
resnext101_32x8d	cpu	88.7913	16.55	79.312	94.526	0.7905506	0.79312	0.7958598
resnext101_32x8d_quantization	cpu	88.7913	16.55	79.072	94.448	0.7877295	0.79042	0.7932688
resnext50_32x4d	cpu	25.0289	4.29	77.618	93.698	0.7732619	0.77618	0.7789037
resnext50_32x4d_quantization	cpu	25.0289	4.29	77.324	93.66	0.7708864	0.77376	0.7770193
shufflenet_v2_x0_5	cpu	1.3668	0.05	60.552	81.746	0.5989082	0.60552	0.6042495
shufflenet_v2_x0_5_quantization	cpu	1.3668	0.05	58.148	79.908	0.5760184	0.58264	0.5843607
shufflenet_v2_x1_0	cpu	2.2786	0.15	69.362	88.316	0.6901337	0.69362	0.696906
shufflenet_v2_x1_0_quantization	cpu	2.2786	0.15	67.784	87.142	0.673757	0.67774	0.6817056
vgg11	cpu	132.8633	7.63	69.02	88.628	0.6854438	0.6902	0.6913732
vgg11_bn	cpu	132.8688	7.65	70.37	89.81	0.6989278	0.7037	0.70702
vgg11_bn_quantization	cpu	132.8688	7.65	70.126	89.59	0.6966344	0.70138	0.7053595
vgg13	cpu	133.0478	11.34	69.93	89.246	0.694549	0.6993	0.7012308
vgg13_bn	cpu	133.0537	11.37	71.586	90.374	0.7112163	0.71586	0.7195748
vgg13_bn_quantization	cpu	133.0537	11.37	71.368	90.242	0.7095961	0.71414	0.7182047
vgg16	cpu	138.3575	15.51	71.592	90.382	0.7116665	0.71592	0.718037
vgg16_bn	cpu	138.366	15.53	73.36	91.516	0.7298424	0.7336	0.7380842
vgg16_bn_quantization	cpu	138.366	15.53	73.182	91.418	0.7283671	0.73198	0.7372297
vgg19	cpu	143.6672	19.67	72.376	90.876	0.7195907	0.72376	0.7255839
vgg19_bn	cpu	143.6782	19.7	74.218	91.842	0.738582	0.74218	0.7460973
vgg19_bn_quantization	cpu	143.6782	19.7	73.984	91.784	0.7365202	0.74002	0.7444082
mobilenetv2	cuda	3.5049	0.31	71.878	90.286	0.7151568	0.71878	0.7213048
pruning_resnet50_0.2_remain	cuda	7.1777	0.94	69.378	89.156	0.689938	0.69378	0.6969834
pruning_resnet50_0.4_remain	cuda	10.3957	1.51	72.912	91.112	0.7259277	0.72912	0.7322989
pruning_resnet50_0.6_remain	cuda	14.5328	2.23	74.58	92.174	0.7427464	0.7458	0.7485525
pruning_resnet50_0.7_remain	cuda	16.9452	2.65	75.204	92.396	0.7494368	0.75204	0.7550905
resnet101	cuda	44.5492	7.87	77.374	93.546	0.7711684	0.77374	0.7778877
resnet152	cuda	60.1928	11.62	78.312	94.046	0.7805866	0.78312	0.7869845
resnet18	cuda	11.6895	1.83	69.758	89.076	0.6929891	0.69758	0.7009789
resnet34	cuda	21.7977	3.68	73.314	91.42	0.7296244	0.73314	0.7368713
resnet50	cuda	25.557	4.14	76.13	92.862	0.7583254	0.7613	0.7655472
resnext101_32x8d	cuda	88.7913	16.55	79.312	94.526	0.7905506	0.79312	0.7958598
resnext50_32x4d	cuda	25.0289	4.29	77.618	93.698	0.7732619	0.77618	0.7789037
shufflenet_v2_x0_5	cuda	1.3668	0.05	60.552	81.746	0.5989082	0.60552	0.6042495
shufflenet_v2_x1_0	cuda	2.2786	0.15	69.362	88.316	0.6901337	0.69362	0.696906
vgg11	cuda	132.8633	7.63	69.02	88.628	0.6854438	0.6902	0.6913732
vgg11_bn	cuda	132.8688	7.65	70.37	89.81	0.6989278	0.7037	0.70702
vgg13	cuda	133.0478	11.34	69.93	89.246	0.694549	0.6993	0.7012308
vgg13_bn	cuda	133.0537	11.37	71.586	90.374	0.7112163	0.71586	0.7195748
vgg16	cuda	138.3575	15.51	71.592	90.382	0.7116665	0.71592	0.718037
vgg16_bn	cuda	138.366	15.53	73.36	91.516	0.7298424	0.7336	0.7380842
vgg19	cuda	143.6672	19.67	72.376	90.876	0.7195907	0.72376	0.7255839
vgg19_bn	cuda	143.6782	19.7	74.218	91.842	0.738582	0.74218	0.7460973

A.2 All model resource and time consumption

model name	device type	CUDA memory footprint(V100 MB)	storage usage(MB)	Host memory usage memory footprint(5k V100 MB)	Inference time(5k images second)	runnig time(5k images unit:second)
mobilenetv2	cpu	0	13.5	11.569882	1.8472979	83.53202
mobilenetv2_quantization	cpu	0	3.63	12.836484	2.620668378	11.49833182
pruning_resnet50_0.2_remain	cpu	0	54.9	61.75387	2.3787837	117.036385
pruning_resnet50_0.4_remain	cpu	0	79.5	138.10234	2.210417	33.568108
pruning_resnet50_0.6_remain	cpu	0	111	184.9352	1.9948468	35.379776
pruning_resnet50_0.7_remain	cpu	0	129	189.98808	2.1977894	101.327515
resnet101	cpu	0	170	275.05652	2.6946783	52.846188
resnet101_quantization	cpu	0	43.23	95.202225	3.424703821	21.65925134
resnet152	cpu	0	230	395.7694	3.2523725	71.65009
resnet152_quantization	cpu	0	58.49	130.00273	5.33926274	26.73317602
resnet18	cpu	0	44.6	68.49211	1.654626	17.790768
resnet18_quantization	cpu	0	11.26	24.66668	1.642623795	13.06700651
resnet34	cpu	0	83.2	93.61113	1.9043267	22.53123
resnet34_quantization	cpu	0	20.96	43.235275	1.866781142	15.35702113
resnet50	cpu	0	97.7	127.908714	2.0651243	38.211304
resnet50_0.2_remain_quantization	cpu	0	7.18	22.005781	2.557384218	42.02664295
resnet50_0.4_remain_quantization	cpu	0	10.27	26.322851	2.603384567	56.28673432
resnet50_0.6_remain_quantization	cpu	0	14.23	32.23258	2.745873715	73.10740106
resnet50_0.7_remain_quantization	cpu	0	16.54	38.593243	2.845771646	83.91774667
resnet50_quantization	cpu	0	24.78	54.308556	2.243978512	16.68061735
resnext101_32x8d	cpu	0	339	534.6753	3.8631282	93.11032
resnext101_32x8d_quantization	cpu	0	86.05	177.23277	4.498770158	38.2392908
resnext50_32x4d	cpu	0	95.7	108.26656	2.1497233	45.882286
resnext50_32x4d_quantization	cpu	0	24.37	50.654373	2.36484581	18.41926982
shufflenet_v2_x0_5	cpu	0	5.3	1.2510937	1.6987754	13.312544
shufflenet_v2_x0_5_quantization	cpu	0	1.44	7.248086	2.728590029	12.73593504
shufflenet_v2_x1_0	cpu	0	8.81	19.117031	1.7465422	26.96716
shufflenet_v2_x1_0_quantization	cpu	0	2.36	13.525937	2.808129019	19.23397367
vgg11	cpu	0	506	518.2353	1.7393055	24.160547
vgg11_bn	cpu	0	506	460.25095	4.132455	277.58932
vgg11_bn_quantization	cpu	0	126.89	135.52383	2.852742979	20.89167565
vgg13	cpu	0	507	518.80457	1.7659446	30.437565
vgg13_bn	cpu	0	507	482.33646	4.268686	275.29953
vgg13_bn_quantization	cpu	0	127.07	135.49684	2.877009756	25.89414021
vgg16	cpu	0	527	539.471	1.8539958	36.121296
vgg16_bn	cpu	0	527	502.75745	3.7599702	41.97046
vgg16_bn_quantization	cpu	0	132.16	145.96101	2.996840029	30.70499454
vgg19	cpu	0	548	559.66534	2.0113995	55.0233
vgg19_bn	cpu	0	548	520.3555	3.8774028	48.31354
vgg19_bn_quantization	cpu	0	137.24	157.19669	3.07490417	35.44899544
mobilenetv2	cuda	930.30505	13.5	2201.809	4.607426	8.776211
pruning_resnet50_0.2_remain	cuda	989.95233	54.9	97.15695	1.3098496	35.722057
pruning_resnet50_0.4_remain	cuda	1028.4344	79.5	134.33293	1.3243407	39.127014
pruning_resnet50_0.6_remain	cuda	1082.3094	111	184.48973	1.3777936	40.769028
pruning_resnet50_0.7_remain	cuda	1115.0192	129	211.79109	1.4713467	44.130047
resnet101	cuda	1088.0818	170	316.1884	2.0004692	45.018383
resnet152	cuda	1138.1085	230	437.7406	2.4627118	61.76
resnet18	cuda	964.93896	44.6	76.110275	1.1651825	15.444048
resnet34	cuda	1007.2693	83.2	135.43117	1.3481721	20.159592
resnet50	cuda	1016.88983	97.7	169.84492	1.5465112	31.291504
resnext101_32x8d	cuda	1243.9343	339	580.1648	2.845158	116.1207
resnext50_32x4d	cuda	1009.1934	95.7	152.72687	1.5501775	52.58519
shufflenet_v2_x0_5	cuda	924.5328	5.3	2189.9666	4.609432	8.639846
shufflenet_v2_x1_0	cuda	926.45685	8.81	2189.732	4.674538	8.649847
vgg11	cuda	1397.8628	506	2188.5906	5.513823	8.794423
vgg11_bn	cuda	1397.8628	506	2186.3591	7.088823	9.025591
vgg13	cuda	1397.8628	507	2189.7734	5.499759	9.005855
vgg13_bn	cuda	1397.8628	507	2186.799	7.114289	9.254742
vgg16	cuda	1417.1038	527	2191.1182	5.5356007	9.124831
vgg16_bn	cuda	1417.1038	527	2186.2605	7.186367	9.366461
vgg19	cuda	1455.5859	548	557.4191	1.3645045	52.017365
vgg19_bn	cuda	1455.5859	548	2187.1748	7.283457	9.486019

A.3 Project Git repository link

GitHub：
https://github.com/bawanag/Comparison-of-Different-Deep-Learning-technologies.git
Main Gitea link:
http://gitea2.taoyuanjian.fun:57281/bawanag/compare_models_and_compressed_tech.git
Backup link:
http://gitea.taoyuanjian.fun:57281/bawanag/choose_your_model_in_EDLS.git