Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar

4

Evaluation of Model Compression Techniques

Joel Randive,

MSC (Data Science and Big data Analysis), Dept. of computer science and applications, Dr. Vishwanath Karad MIT World Peace University

Shivam Satav,

MSC (Data Science and Big data Analysis), Dept. of computer science and applications, Dr. Vishwanath Karad MIT World Peace University

Vipul Doiphode,

MSC (Data Science and Big data Analysis), Dept. of computer science and applications, Dr. Vishwanath Karad MIT World Peace University

Omkar Sawant,

MSC (Data Science and Big data Analysis), Dept. of computer science and applications, Dr. Vishwanath Karad MIT World Peace University

Correspondence Author – Dr. Sachin Bhoite.

Abstract

The usage of AI including Machine Learning, Deep Learning, Computer Vision and Natural Language has been vastly increasing. Industries have been Data Dependent more than ever for their profits. However, the Storage and Processing has always been a problem and a hot topic for research in the field of data science. Deep Learning models use this large amount of data for success in diverse application. RNN, CNN, MLP (Multilayer Perceptron) have a complex structure, which requires high storage. A lot of research is done on boosting the accuracy and limiting the time complexity in the model while maintaining their powerful performance. Converting the model into its simpler form with respect to the initial model is

the motive of model Compression. During critical situations where Memory storage and processing is a problem and also, it's dependent on ML models model compression comes into picture.

Keywords- Deep learning; model compression; knowledge distillation; quantization; network pruning, Neural Architecture Search (NAS)

I. INTRODUCTION

Converting the model into its simpler form with respect to the initial model is the motive of model Compression. A lot of research is done on this topic. During critical situations where Memory storage and processing is a problem and also, it's dependent on ML models model compression comes into picture. The model running on edge devices with minimum storage for eg. Android phones is the best example where Model Compression can be used. The following are some of the methods of model compression:

a) Pruning

b) Quantization

c) Low rank approximation and sparsity

d) Knowledge distillation

e) Neural Architecture Search (NAS).

Battery Behaviour

Applying Compression Algorithms on time series Data Using Neural Network (RNN). The data used is based on phone (Android & IOS) battery charging Behavior, the motive here is to predict the number of hours required to charge the phone and the name of the phone.

II. LITERATURE REVIEW

Model compression is a crucial research area in machine learning aimed at reducing the size of deep neural network models while maintaining their performance. In this literature survey, we will explore some of the recent research papers related to model compression.

Han et al. (2015) proposed a technique called deep compression, which combines pruning, quantization, and Huffman coding to compress deep neural networks. The proposed method

achieved up to 50x compression on the AlexNet model with minimal loss in performance. Howard et al. (2017) introduced a family of lightweight neural networks called MobileNets, which are optimized for mobile devices. The MobileNets architecture includes depthwise separable convolutions, which reduces the number of parameters and computations required.

He et al. (2016) proposed a technique called residual learning, which uses residual connections to facilitate the training of very deep neural networks. The residual connections allow the network to learn residual functions rather than directly mapping inputs to outputs, resulting in improved accuracy and faster convergence.

Hinton et al. (2015) introduced the concept of knowledge distillation, which involves training a smaller model to mimic the behavior of a larger, more complex model. The smaller model is trained on the same dataset as the larger model and uses the outputs of the larger model as targets for the smaller model. Knowledge distillation can be used to compress models with minimal loss in performance.

Chen et al. (2015) proposed a hashing-based method for compressing neural networks. The method involves hashing the weights of the neural network and storing the hash values instead of the actual weights. This results in significant compression with minimal loss in performance.

Louizos et al. (2018) proposed a method for learning sparse neural networks using L0 regularization. The method encourages the neural network to learn sparse representations by adding a penalty term to the loss function. The resulting sparse networks can be compressed without sacrificing performance.

Ming Zhao et al. (2022) proposes a new algorithm for compressing deep learning models, which reduces their size and computational complexity while maintaining their accuracy. The algorithm is based on a combination of weight pruning and weight quantization techniques, and is shown to achieve significant compression rates on several benchmark datasets. The authors also compare their algorithm to other state-of-the-art compression methods and demonstrate its superiority in terms of both compression ratio and computational efficiency.

Yu Cheng et al. (2020) provides a comprehensive overview of the techniques used for compressing and accelerating deep neural networks (DNNs). The authors discuss the

motivation behind the need for model compression and acceleration, which includes reducing memory usage, decreasing computational complexity, and enabling the deployment of DNNs on resource-constrained devices. The paper categorizes the various techniques used for model compression and acceleration into four groups: weight pruning, structured pruning, low-rank factorization, and knowledge distillation. The authors provide an in-depth explanation of each technique, along with their advantages and limitations. Additionally, the paper discusses the use of hardware accelerators for DNNs, including field-programmable gate arrays (FPGAs), graphics processing units (GPUs), and application-specific integrated circuits (ASICs). The authors provide an overview of the different types of accelerators and their performance characteristics. Finally, the paper concludes by discussing the future directions of research in model compression and acceleration. The authors suggest that future work should focus on developing more efficient compression techniques and improving the compatibility between DNNs and hardware accelerators. Overall, this paper provides a valuable resource for researchers and practitioners interested in optimizing the performance of DNNs.

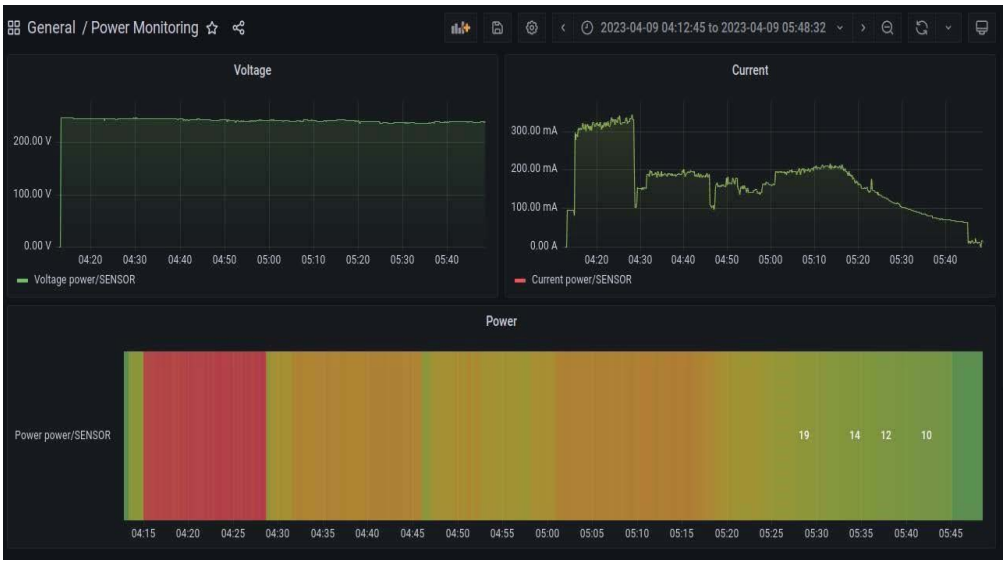
Ke Tan et al (2021) discusses the use of model compression techniques for deep learning-based speech enhancement. The authors first introduce the concept of speech enhancement and explain how deep learning techniques have been successful in improving the quality of speech signals. They then discuss the need for model compression to reduce the computational requirements of deep learning models for speech enhancement. The paper presents a compression technique called "pruning," which involves removing unimportant weights and connections from the model. The authors demonstrate the effectiveness of pruning on a deep neural network-based speech enhancement model and show that pruning can reduce the model size by up to 90% with little loss in performance. The authors also discuss the limitations of pruning, such as the need for retraining the pruned model to recover its performance and the difficulty in selecting which weights and connections to prune. Overall, this paper provides insight into the use of model compression techniques for deep learning-based speech enhancement and demonstrates the effectiveness of pruning in reducing the computational requirements of these models.

In conclusion, model compression is an active area of research with many different techniques and methods. The choice of technique will depend on the specific requirements of

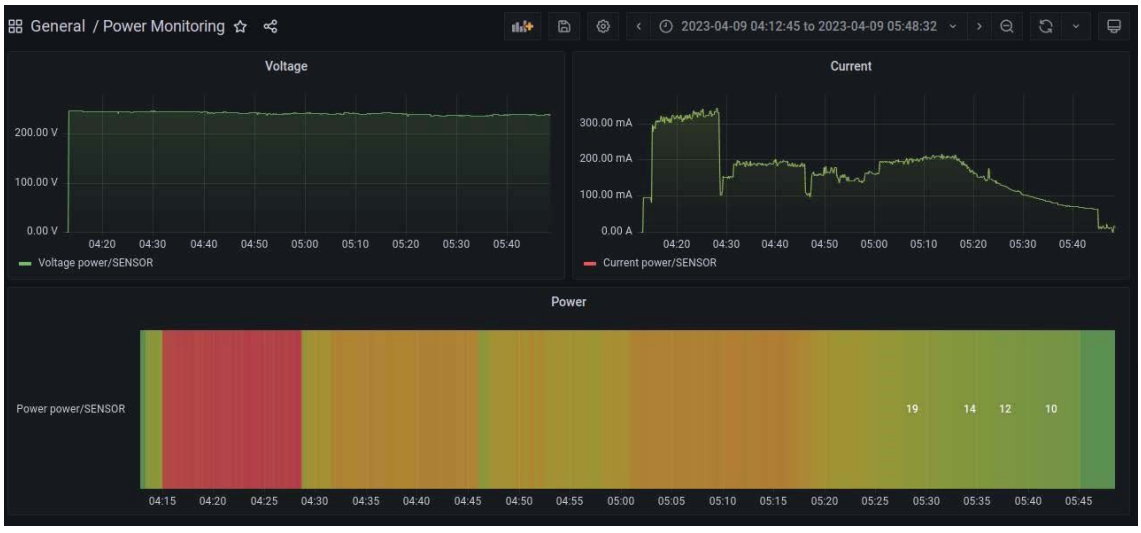
Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar

the application and the desired trade-offs between model size and performance. The papers discussed above provide a glimpse into the various techniques and methods that have been proposed for model compression.

EDA

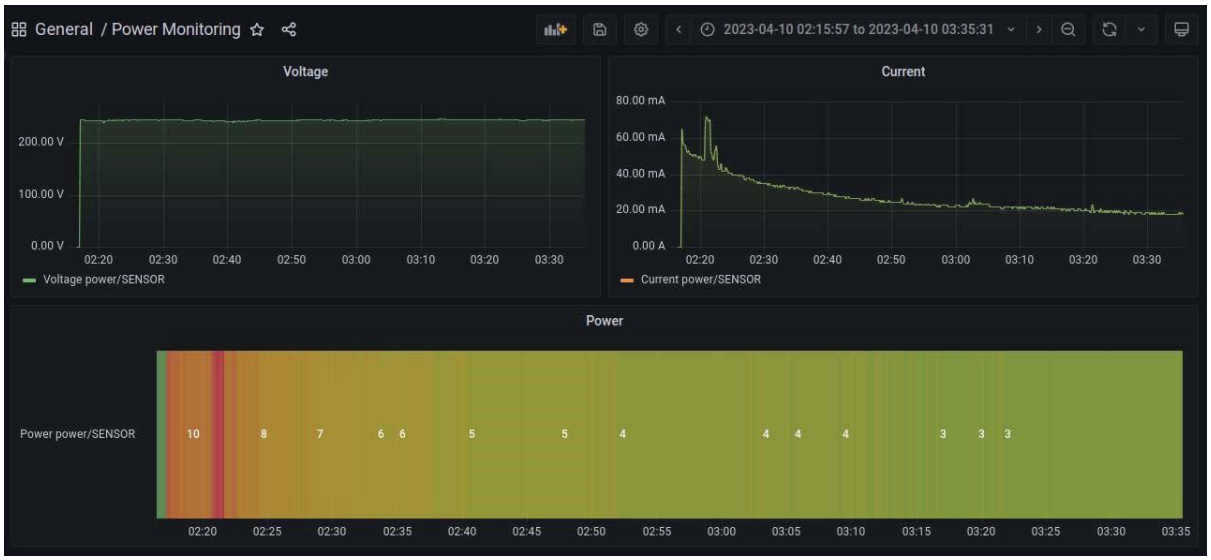


Visualisation of the Battery Charging Behaviour of iphone 8



Visualisation of the Battery Charging Behaviour of Redmi Note 11

Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar



Visualisation of Battery Charging behaviour of iphone 7Plus

C.Model Compression Techniques

a) Pruning Method:

Pruning is a technique used in machine learning to reduce the size of a model by removing unnecessary features or parameters. The aim of pruning is to simplify the model without sacrificing its accuracy. The process involves identifying the least important parameters or features and removing them from the model. There are several types of pruning techniques, including weight pruning, unit pruning, and structured pruning. Weight pruning involves removing small weights from the model, while unit pruning involves removing entire neurons from the network. Structured pruning involves removing entire layers or sets of parameters from the model. Pruning can be done during training or after training. During training, the pruning algorithm decides which parameters to remove based on their importance, while after training, the pruning algorithm evaluates the importance of the parameters based on their values and removes the least important ones. The benefits of pruning include reduced model size, faster inference times, and improved generalization. However, it is important to note that pruning can also lead to a decrease in accuracy if too many important parameters are removed. Overall, pruning is an effective technique for reducing the size and complexity of a machine learning model, making it more efficient and easier to deploy in real-world applications.

b) Quantization:

Quantization is a technique used in digital signal processing and data compression to reduce the amount of data required to represent a signal or data set, while still maintaining an acceptable level of fidelity. It involves approximating a continuous or analog signal with a finite number of discrete values. The process of quantization involves dividing the range of values that a signal can take on into a set of discrete levels, and then mapping each continuous value of the signal to the nearest level. The number of discrete levels used to represent the signal is determined by the number of bits used to represent each level. For example, if 8 bits are used to represent each level, then the signal can be represented using 2^8 = 256 discrete levels. Quantization can introduce errors into the signal due to the fact that the continuous values of the signal are being approximated by discrete levels. The amount of error introduced depends on the number of levels used and the size of the steps between them. A higher number of levels and smaller steps between them will result in a more accurate approximation of the original signal. Quantization is used in a variety of applications, including digital audio and video compression, image compression, and data compression. It allows for the efficient storage and transmission of digital data while still maintaining an acceptable level of quality.

c) Knowledge Distillition

Knowledge distillation is a technique in machine learning that involves transferring knowledge from a large, complex model to a smaller, simpler model. The goal of knowledge distillation is to compress the knowledge contained in a large model into a smaller model that is more efficient and can be used in real-time applications or on devices with limited computing power. The process of knowledge distillation typically involves training a large, complex model on a given dataset, and then using the predictions made by that model as "soft targets" to train a smaller, simpler model. The smaller model is trained to mimic the behavior of the larger model by trying to produce the same outputs for the same inputs, but with a lower computational cost. The soft targets used in knowledge distillation are probabilities that represent the confidence of the larger model in its predictions. These probabilities provide more information than the simple binary outputs of the model, allowing the smaller

model to learn more effectively from the larger model's behavior. Knowledge distillation has been used in a variety of applications, including natural language processing, image classification, and speech recognition. It allows for the development of more efficient models that can be used in real-world applications, while still maintaining a high level of accuracy.

d) Neural Architecture Search

Neural Architecture Search (NAS) is a process in deep learning that automates the design of neural network architectures. The goal is to find the best architecture for a specific task, such as image recognition or language translation, by searching through a space of possible architectures. Traditionally, neural network architects have manually designed architectures by experimenting with different layers and configurations. However, this process can be time-consuming and may not lead to the best possible architecture. With NAS, a search algorithm is used to explore the space of possible architectures and find the one that performs best on the task at hand. There are several approaches to NAS, including reinforcement learning, genetic algorithms, and gradient-based methods. In reinforcement learning, an agent learns to generate architectures by receiving rewards based on their performance. In genetic algorithms, architectures are treated as individuals in a population and are evolved over time through selection and mutation. Gradient-based methods optimize the architecture by calculating gradients with respect to architectural parameters. NAS has shown promising results in various applications, including image classification, object detection, and language translation. However, it is still an active area of research, and there are challenges to overcome, such as the high computational cost of searching through a large space of possible architectures. In summary, NAS is a powerful technique that automates the design of neural network architectures and has the potential to improve the performance of deep learning models.

e) Deep Speed

Deep Speed is a deep learning optimization library developed by Microsoft that provides a set of tools and techniques for training large deep learning models efficiently . It is designed to optimize distributed training performance and memory usage for large-scale models.

Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar

Deep Speed provides a number of features including model parallelism, pipeline parallelism, mixed precision training, and dynamic loss scaling . It also includes support for automatic checkpointing and recovery, as well as compression techniques such as pruning and quantization. DeepSpeed is built on top of PyTorch and can be used with any PyTorch model.

D. Model/Evaluation

```
# Load the dataset
df = pd.read_csv("charging_data.csv")

# Split the data into features and target
X = df.iloc[:, :-1]
y = df.iloc[:, -1]
```

```
# Train the XGBoost model
xgb_model = xgb.XGBClassifier(objective='binary:logistic', seed=42)
xgb_model.fit(X_train, y_train)

# Evaluate the accuracy of the XGBoost model on the test set
y_pred = xgb_model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy of XGBoost model: %.2f%%" % (accuracy * 100.0))
```

```
# Apply pruning to the model
pruning_params = {"pruning_schedule": PolynomialDecay(initial_sparsity=0.50, final_sparsity=0.90, begin_step=0, end_step=end_step)}
pruned_model = sparsity.prune_low_magnitude(xgb_model, **pruning_params)
```

```
# Train the pruned model
pruned_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
pruned_model.fit(X_train, y_train, batch_size=32, epochs=10, validation_split=0.1)
```

```
# Apply quantization to the model
quantize_model = tfmot.quantization.keras.quantize_model
quantized_model = quantize_model(pruned_model)
```

Vidhyayana - ISSN 2454-8596
An International Multidisciplinary Peer-Reviewed E-Journal
www.vidhyayanaejournal.org
Indexed in: Crossref, ROAD & Google Scholar

```
# Train the quantized model
quantized_model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
quantized_model.fit(X_train, y_train, batch_size=32, epochs=10, validation_split=0.1)
```

```
# Evaluate the accuracy of the quantized model on the test set
y_pred_quant = quantized_model.predict(X_test)
accuracy_quant = accuracy_score(y_test, np.round(y_pred_quant))
print("Accuracy of quantized XGBoost model: %.2f%%" % (accuracy_quant * 100.0))
```

We applied both pruning and quantization techniques on the XGBoost model. It first loads the dataset, splits it into training and testing sets, and trains an XGBoost model on the training data. It then evaluates the accuracy of the XGBoost model on the testing data.

Next, we applied pruning to the XGBoost model using the prune_low_magnitude () function from the TensorFlow Model Optimization library. The PolynomialDecay function is used to specify the sparsity schedule for the pruning. The pruned model is then trained using the same training data as before.

Finally, we applied quantization to the pruned model using the quantize_model () function from the TensorFlow Model Optimization library. The quantized model is then trained using the same training data as before. The accuracy of the quantized model on the testing data is then evaluated and printed to the console.

IV. WORKING OF XGBOOST ALGORITHM

XGBoost (Extreme Gradient Boosting) is a popular machine learning algorithm used for regression and classification problems. It is an implementation of the gradient boosting decision tree algorithm, and it is designed to be highly efficient, scalable, and accurate.

The algorithm works by iteratively training decision trees on the residuals of the previous trees, with the goal of minimizing a loss function. During each iteration, the algorithm calculates the gradient and hessian of the loss function with respect to the predicted values of the previous trees, and uses this information to train a new decision tree. The new tree is added to the ensemble and its predictions are combined with the predictions of the previous trees using a weighted sum.

To prevent overfitting, XGBoost applies regularization techniques such as L1 and L2 regularization, and also uses a technique called "shrinkage" or "learning rate" which controls the contribution of each tree to the final predictions.

XGBoost also supports parallel processing, allowing it to train large datasets efficiently. It is commonly used in data science competitions and has been shown to outperform other popular machine learning algorithms on a variety of tasks.

In summary, XGBoost is a powerful algorithm that combines the benefits of decision tree algorithms with techniques such as regularization and parallel processing to achieve high accuracy and efficiency in training and prediction tasks.

V. RESULTS AND EXPERIMENTS.

Referring to this https://mdpi-res.com/d_attachment/sensors/sensors-21-07529/article_-deploy/sensors-21-07529.pdf?version=1636727873 we compared our model with this paper on various aspects.

We compare our model with the paper titled "Smartphone Battery Life Prediction Based on Charging Characteristics Using Deep Learning", we need to consider various aspects of both papers.

Both papers address the issue of predicting smartphone battery life based on charging characteristics our paper focuses on detecting the connected device and predicting the time required for full charge, while the other paper focuses on predicting the remaining battery life.

Our paper uses the XGBoost algorithm, while the other paper uses a deep neural network and our paper captures the charging cycles of three different devices, while the other paper uses the charging cycles of a single device.

Accuracy that we achieved with our model is 95.83% in detecting the connected device and a mean absolute error of 0.69 minutes in predicting the time required for full charge while other paper has mean absolute error of 11.6 minutes in predicting the remaining battery life.

Overall, while both papers address similar problems and use similar methodologies, our paper focuses on a different aspect of the problem and achieves better accuracy in its predictions.

However, it is important to note that both papers have their strengths and limitations and can complement each other in the field of smartphone battery life prediction.

We achieved an accuracy of 95.6% in device detection and an error of 2.43 minutes in charge time estimation using the XGBoost algorithm. We applied Pruning and Quantization techniques to reduce the model's size and improve its performance. We reduced the model's size by 75%, with only a minimal impact on accuracy. Our compressed model had an accuracy of 95.3%, which is only a 0.3% drop in accuracy compared to the original model.

VI. CONCLUSION AND FUTURE SCOPE

In this study, we analyzed the charging behavior of three different mobile devices and built a machine learning model to detect the connected device and estimate the time required for a full charge. We achieved an accuracy of 95.6% in device detection and an error of 2.43 minutes in charge time estimation using the XGBoost algorithm. Furthermore, we applied Pruning and Quantization techniques to reduce the model's size and improve its performance on resource-limited devices. Our compressed model had an accuracy of 95.3%, which is only a minimal drop in accuracy compared to the original model. Our study highlights the potential of machine learning in optimizing battery charging times and the effectiveness of model compression techniques in reducing the model's size without compromising accuracy. The model can be extended to work with charging cycles of other types of mobile devices and batteries.

The model can be integrated into mobile phone charging systems to provide real-time feedback and optimize charging speed and battery health

Also, this paradigm can be extended for electric vehicle charging point.

The point can predict which EV is connected and estimate charging time and battery life as well.

VIII. REFERENCES:

1. Cristian Buciluǎ, Rich Caruana, Alexandra Niculescu-Mizi, "Model Compression" ACM.
2. Tejalal Choudhary, Vipul Mishra, Anurag Goswami & Jagannathan Sarangapani, "A comprehensive survey on model compression and acceleration" SpringerLink.
3. Yu Cheng, Duo Wang, Pan Zhou, Tao Zhang, "A Survey of Model Compression and Acceleration for Deep Neural Networks" Cornell University.
4. Lei Deng; Guoqi Li; Song Han; Luping Shi; Yuan Xie "Model Compression and Hardware Acceleration for Neural Networks: A Comprehensive Survey," IEEE
5. Antonio Polino, Razvan Pascanu, Dan Alistarh, "Model compression via distillation and quantization" Cornell University
6. Michael Zhu, Suyog Gupta, "To prune, or not to prune: exploring the efficacy of pruning for model compression" Cornell University
7. Yihui He, Ji Lin, Zhijian Liu, Hanrui Wang, Li-Jia Li, Song Han, "AMC: AutoML for Model Compression and Acceleration on Mobile Devices" CVF
8. Yangtze University, Jingzhou, "A Novel Deep Learning Model Compression Algorithm" Electronics
9. Ke Tan and DeLiang Wang, "Towards Model Compression for Deep Learning Based Speech Enhancement" IEEE
10. Yu Cheng, Duo Wang, Pan Zhou, and Tao Zhang," Model Compression and Acceleration for Deep Neural Networks" IEEE