Efficient Checkpointing for Deep Neural Networks

Bryan Chen, bryan.chen@etu.toulouse-inp.fr[a], t0934135@u.nus.edu[b]

[a]*Polytechnic National Institute, ENSEEIHT, Toulouse, France,*
[b]*National University of Singapore, Singapore*

Degree of Engineer in Computer Science
High-Performance Computing, Big Data, and Artificial Intelligence

14/03/2024 - 14/09/2024

University tutor
Olivier Cots
olivier.cots@enseeiht.fr

Company tutor
Wei Tsang Ooi
ooiwt@comp.nus.edu.sg

ACKNOWLEDGEMENTS

"Alone, we can do so little; together, we can do so much"
- Helen Keller

The research internship was transformative, thanks to the constant guidance, invaluable support, and extensive knowledge of mentors Mr. Ooi Wei Tsang (NUS), Mr. Yannis Montreuil (Sorbonne, UPMC), Mr. Axel Carlier (INP-ENSEEIHT), and Mr. Ng Lai Xing (A*STAR). Their expertise and availability not only enriched my academic experience but also brought fresh and innovative perspectives to my research work. I am particularly grateful for the stimulating discussions and valuable guidance they provided throughout this period.

I would also like to express my gratitude to the entire team at CNRS@Create for their collaborative efforts and camaraderie. The dynamism and team spirit prevalent within this organization have greatly contributed to the success of my internship. Exchanges with my colleagues, always ready to share their ideas and experiences, were invaluable in overcoming challenges and advancing my projects.

A special thank you to my friends and colleagues at CNRS@Create, whose solidarity and support have made this experience not only rewarding from a professional standpoint but also enjoyable and memorable on a personal level. Moments of conviviality and informal discussions were just as important as the work sessions and have forged bonds that I hope will last well beyond the end of this internship.

Lastly, I wish to thank all those who, directly or indirectly, contributed to the success of this experience. Their encouragement and support have been key elements of my motivation and commitment throughout this journey.

Abstract

Re-training all the accumulated data since the occurrence of data poisoning to obtain a new, up-to-date model for deployment, particularly in the context of critical systems, is often challenging. Thus, a branch and a training of the model on potentially corrupted data and definitively uncorrupted data can be created from the outset. This means that almost immediate recovery time could be achieved, as an uncorrupted branch can be directly used in the event of data poisoning. Additionally, a branch of models trained on potentially corrupted data could allow for further investigations and studies on the source and impacts of the corrupted data. Although a short recovery time is achieved, a large number of models must be trained and stored. Consequently, an appropriate checkpoint and compression system is necessary. The goal of this study is to create a checkpoint mechanism for the model training process in the context of "branching" – focusing on achieving the best compression performance with acceptable restoration accuracy.

Keywords: Machine Unlearning; Optimization; Compression; Quantization; Low-rank Adaptation; Checkpoints

Contents

List of Figures

List of Tables

1 Context

1.1 Introduction to CNRS@Create

CNRS@Create [1], established in 2019, is the first subsidiary of CNRS based in Singapore, located within the CREATE (Campus for Research Excellence and Technological Enterprise). CREATE was founded by the National Research Foundation (NRF) [2] in 2006 to enhance the vitality and diversity of Singapore's research and development (R&D) ecosystem. CREATE brings together researchers from ten institutions in collaboration with the National University of Singapore (NUS), Nanyang Technological University (NTU), and A*STAR (Agency for Science, Technology and Research) across fourteen interdisciplinary research programs, including:

- Massachusetts Institute of Technology (MIT)
- University of California Berkeley (UCB)
- Cambridge University
- ETH Zurich
- Technical University of Munich (TUM)
- Hebrew University of Jerusalem
- Centre National de la Recherche Scientnifique (CNRS)
- University of Illinois at Urbana-Champaign (UIUC)
- Imperial College London (ICL)
- Shanghai Jiao Tong University (SJTU)

CNRS@Create represents a total investment of 50 million Singapore dollars, half of which is provided by the Singaporean government and the rest by CNRS. Several programs are developed here, such as Space (2 years), Calypso (2 years), ScanCells (2 years), EcoCTs (2 years), and Descartes (5 years). CNRS@Create aims to:

- Strengthen the global position of France and Singapore in research fields with the greatest potential for present and future society.
- Create unique initiatives in transdisciplinary research excellence and technological development that would not exist under normal circumstances or within a single country or institution.
- Support translational research and innovation to promote the exploitation of research results. Direct industry participation in research projects is encouraged to ensure an effective transformation of research outcomes into innovative products.

1.2 Introduction to the Descartes Program

The Descartes program consists of ten "work packages" (WP), divided into three pillars, with a global team of 80 permanent researchers, 29 PhD students, and sev-

eral engineers from either academic members (CNRS, ENSAM, ENS Paris-Saclay, Sorbonne University, Grenoble INP, Université Côte d'Azur, Université Toulouse III, Université Lyon III, UGA, ENAC, Université PSL, ENSEEIHT, Université de Bordeaux, Université de Strasbourg, Université Laval, Universidad Zaragoza, Eindhoven University of Technology – TU/e) or academic members from Singapore (NUS, NTU, A*STAR, SUSS).

The Descartes program aims to develop disruptive hybrid artificial intelligence to serve the smart city (like a digital twin of Singapore cf. FIGURE 1.1) and enable optimized decision-making in complex situations encountered by critical urban systems, covering several domains from social sciences and virtual reality to intelligent use of AI.

Figure 1.1: DesCartes Augmented Marina Bay Twin

The program is structured into three pillars:

1. **Intelligent Computing:** Develop computational tools to manage smart data and produce certified and explainable hybrid twins and controllers, based on physical models and knowledge.
2. **Empowering People in Smart Societies:** Study the collaborative decision-

making process between human operators and hybrid AI, develop natural language interfaces for more natural interactions between operators and hybrid AI, draft ethical guidelines for policy makers and hybrid AI designers, propose a minimal set of guidelines for legal regulation and policy development for hybrid AI implementation, and explore global integration and co-development processes of Descartes.

3. **Engineering and Builders:** Bring engineering, technological, and application dimensions to the concept of hybrid AI. It aims to propose an operational methodology and architecture, from intelligent sensing and predictive diagnostics to robust and scalable control, for complex and critical systems.

The program focuses on five main use cases, with the collaboration of several industries (ESI Group, CETIM-MATCOR, EDF, ARIA Technologies, SKF, Thales, Immersion, Naval Group, and Expleo) and partners (DSO National Laboratories, Azur Drones, Singapore Institute of Technologies - SIT):

1. Study of wind maps
2. Planning of drone trajectories
3. Remote data collection
4. Searching for alternatives in emergencies
5. Optimization and prediction of the energy network

1.3 Description of work package 4

Work Package 4 (WP4) falls under the "Empowering People in Smart Societies" pillar of the Descartes program and consists of approximately fifteen scientists, including five from Singapore and four from France. It focuses on how humans can interact with AI to:

1. Bring human aspects that cannot be modeled computationally into AI systems and algorithms, forming a hybrid AI with human interaction at its core.
2. Enable hybrid AI to enhance human perception and cognition (notably assisting humans in decision-making).

Within this WP, they are developing interaction and visualization techniques for collaboration between humans and AI to build intelligent, explainable, and accountable AI, also exploring how to enhance the robustness of AI against erroneous and malicious human inputs. Finally, they will examine how hybrid AI can augment human perception and cognition. WP4 thus includes two main tasks:

1. **Learning to defer**: Involving machine learning models that make autonomous decisions based on their confidence and solicit experts when necessary to improve the model.
2. **Unlearning from experts**: In cases of malicious or biased data, detection of

such data is taken for granted, and the issue is how to efficiently remove this data from our model.

2 Work Organization

The internship was part of the Descartes project at CNRS@Create, under the supervision of Mr. Ooi Wei Tsang (NUS), Mr. Axel Carlier (INP-ENSEEIHT), Mr. Lai Xing Ng (A*STAR), and Mr. Yannis Montreuil (Sorbonne Université, UPMC).

On one hand, PhD Yannis Montreuil worked on the topic "Learning to defer", supervised by the Lead Principal Investigator (PI) Mr. Ooi Wei Tsang, and PIs Mr. Axel Carlier and Mr. Lai Xing Ng. His work involved proposing methods that integrate expert knowledge into AI algorithms to enhance decision-making. In this collaborative system, predictions are deferred to a qualified expert when the model encounters uncertainties or errors. This work lies at the intersection of statistics, machine learning, and social sciences.

On the other hand, the topic of my internship complemented the work of Mr. Yannis Montreuil when the expert provides malicious or biased data, thus it was necessary to develop an effective machine learning model training framework that addresses this problem. The proposed solution was to integrate a system of compressed checkpoints to forget the malicious data, assuming detection is granted.

The work during this internship was structured to balance theoretical research and practical applications. Each week, specific objectives were set to ensure continuous and coherent progress. Weekly calls were organized to discuss advancements, challenges encountered, and solutions considered.

At the beginning of the internship, as my supervisor had already assigned the task to a NUS student to implement this system, I started by revisiting the work of this student. This involved assimilating and understanding his code before developing and improving upon his work. This initial phase was crucial for establishing a solid foundation on which I could build and suggest improvements. Moreover, it also involved reading and understanding the research papers associated with the techniques involved, notably LC-checkpoint and Delta-LoRA, which is a modified version of LoRA, thus understanding LoRA was also essential to grasp Delta-LoRA and the schema used in the project.

Each week, meetings were held with Mr. Ooi Wei Tsang, Mr. Yannis Montreuil, Mr. Axel Carlier, and Mr. Lai Xing Ng, to share progress made, receive feedback, and adjust goals based on the challenges encountered. This structured framework allowed for ongoing progress and ensured that the project's objectives were achieved

in an efficient and organized manner.

3 Work Methods

The work method adopted during this internship relied on a systematic and iterative approach to ensure methodical progression and quality results. This approach was structured around the following elements:

3.1 Preliminary Analysis and Understanding of Context

The first step involved conducting a thorough analysis of the project needs and understanding the scientific and technical context. This included:

- Reading and rereading relevant research articles, particularly those focusing on LC-checkpoint, Delta-LoRA, and LoRA.
- Understanding the underlying theoretical concepts and schemes used in these techniques.
- Analyzing previous work, particularly the code and contributions of the NUS student, to grasp the fundamentals and specific implementations.

3.2 Planning and Setting Goals

Based on the preliminary analysis, a detailed work plan was developed. This plan included:

- Defining specific, achievable weekly goals to maintain continuous progression.
- Creating a schedule for weekly meetings to discuss progress, challenges, and necessary adjustments.
- Establishing intermediate milestones to track progress and adjust priorities accordingly.

3.3 Development and Implementation

The next step involved the development and implementation of technical solutions. This included:

- Assimilating and improving the existing code provided by the NUS student.
- Implementing new features and enhancing existing algorithms to meet the specific needs of the project.
- Establishing a system of compressed checkpoints to handle malicious or biased data provided by experts.

3.4 Testing and Validation

To ensure the quality and effectiveness of the developed solutions, a rigorous testing and validation phase was conducted. This phase included:

- Designing unit tests to verify the functionality and reliability of the code.
- Analyzing test results to identify and correct any potential errors or inefficiencies.

3.5 Documentation and Communication

Finally, comprehensive and clear documentation was prepared to ensure traceability and reproducibility of the work. This documentation included:

- Writing detailed reports on the methodologies used, results obtained, and conclusions drawn.
- Creating user guides and technical manuals to facilitate understanding and usage of the developed tools.
- Regularly communicating progress and results to supervisors and team members through meetings and presentations.

This structured work method allowed the project to be successfully completed efficiently, ensuring continuous progress and quality results, while fostering active and productive collaboration within the team.

4 Work Tools

To successfully complete the tasks and objectives of the internship, a suitable selection of tools and technologies was utilized. These tools played a crucial role in managing, developing, and evaluating the proposed solutions. Here are the main tools used:

4.1 Integrated Development Environment (IDE)

The IDE chosen for code development was crucial for ensuring maximum productivity and efficiency. I primarily used:

- **Visual Studio Code:** Used for specific tasks requiring increased flexibility and lightness, as well as for editing Markdown files and managing documentation.

4.2 Version Control Tools

To efficiently manage source code and facilitate collaboration with the research team, version control tools were used:

- **Git:** Used with private GitHub repositories to version code, track changes, manage development branches, and facilitate collaboration with other team members.
- **GitHub:** The platform used to host Git repositories, manage issues, and facilitate discussions and code reviews within the team.

4.3 Libraries and Frameworks

For the development of algorithms and data manipulation, several libraries and frameworks were essential:

- **NumPy:** For efficient data manipulation and numerical computation in Python.
- **PyTorch:** For implementing and training deep learning models.
- **Hugging Face Transformers:** For working with pretrained models and developing advanced algorithms using transformers.

4.4 Virtualization and Package Management Environment

To maintain a clean development environment and manage software dependencies, the following tools were used:

- **Conda:** Used for managing data environments and installing specific packages necessary for the project.

4.5 Communication and Collaboration Tools

To facilitate communication and collaboration with supervisors and team members, several tools were used:

- **Zoom:** For weekly virtual meetings, discussions, and updates on project progress.
- **Slack:** Used for instant communications, file management, and quick information sharing within the team.
- **Google Slides:** For creating detailed presentations of results at meetings and conferences, allowing for clear and interactive visualization of project progress and discoveries.

4.6 Supercomputers and Hardware Resources

For the intensive computations necessary for training machine learning models, several hardware resources were made available:

- **ASPIRE 2A supercomputers from NSCC:** Comprised of 4 NVIDIA A100 40G SXM and AMD EPYC 7713, with communication via PBS for server interaction.

- **VPN at NUS:** Used for securely connecting to the computing server.
- **CNRS@Create Computer:** Equipped with two NVIDIA GeForce RTX 3090 for additional computations.
- **Personal Computer:** Equipped with a 12th generation Intel(R) UHD Graphics i7 and a NVIDIA GeForce RTX 4060 Laptop GPU for local development and testing.
- **PuTTY:** Used for connecting to the NSCC server.
- **PBS (Portable Batch System) Commands:** Used to launch .sh tasks on the NSCC server.
- **FileZilla:** Used to transfer files from my local computer to the NSCC server.

4.7 Visualization and Monitoring of Training

For the visualization of neural network training metrics:

- **TensorBoard:** Initially used for visualizing training metrics.
- **Weights Biases (wandb):** Later chosen for its ease of use and intuitive user interface, facilitating the monitoring and comparison of experiments, and preferred over TensorBoard for a better user experience.

4.8 Data Analysis and Visualization Tools

To compare multiple configurations of machine learning models, particularly Vision Transformers (ViT), the following tools were used:

- **R (ggplot2 library):** Used to create graphs and compare different configurations of machine learning models, allowing for a clear and detailed visual analysis of performance.

These tools played a crucial role in efficiently managing the project, facilitating collaboration, ensuring traceability of developments, and enabling agile management of the development and evaluation processes.

5 Introduction

5.1 Internship Topic

The internship focuses on WP4, which deals with robust Human-AI collaboration. The goal is to integrate human experts into the learning loop when the AI model lacks precision. In cases where human bias is detected or malicious data is encountered, the aim is for the AI to "unlearn" from this data, an approach called "unlearning from experts."

The naive approach would be to relearn everything from scratch, but this would be costly in terms of storage space, computational cost, and therefore time.

A less expensive approach, which is the focus of the internship, involves techniques for saving compressed checkpoints during the model training. When malicious data is detected during training, the idea is to revert to the checkpoint (branch) before the occurrence of the detection, and then continue training from this branch. To compress the checkpoints, the idea of the internship is to combine two complementary algorithms, compressing these checkpoints, thus making the saving of these checkpoints less costly in terms of storage and time, as opposed to the idea of recording all model weights for each branch.

Concretely, the approach involves applying a first algorithm, LC-checkpoint [3], which consists of exponential-based quantization followed by priority promotion and then lossless information compression via Huffman encoding. This is combined with the second algorithm, Delta Low-Rank Adaptation (Delta-LoRA) [4], to further reduce the size of the checkpoints. These algorithms will be described in more detail in the following subsection.

The idea of the internship is thus to verify the relevance of combining these two techniques. A preliminary step would be to perform a Proof of Concept (POC) on simple image classification models: LeNet [5], AlexNet [6], and a Lite Version of VGG16 [7], and then to extend this to more costly, newer models such as the Full Version of VGG16, ResNet-50 [8], and Vision Transformer (ViT) [9]. It also involves conducting these verifications on increasingly large and complex databases, starting with MNIST [10], then expanding to CIFAR-10 [11], possibly CIFAR-100 [12], and ImageNet [13].

Subsequently, if the results prove relevant and if time permits, the idea would also be to develop an API accessible to researchers at CNRS@Create so that other WPs of the Descartes program can use this technology to train their machine learning models.

5.2 Description of the Techniques Used

In the context of this research internship, an innovative approach is proposed to optimize the compression of checkpoints by combining two advanced techniques: LC-checkpoint (Lossy Compression Checkpoint) and Delta-LoRA. Before detailing this hybrid method, it is essential to understand individually how each algorithm functions.

5.2.1 LC-checkpoint

5.2.1.1 Introduction

The architecture of the LC-checkpoint [14], as introduced in the research article "On Efficient Constructions of Checkpoints" published in 2020 by Yu Chen, Zhenming Liu, Bin Ren, and Xin Jin, aims primarily to provide an efficient solution for optimizing the management of checkpoints in computer systems in the event of a failure (such as gradient explosion [15] and division by zero [16]) and for saving intermediate states. This section explains the architecture of LC-checkpoint, the motivation behind this method, its utility, and its operation.

5.2.1.2 Motivation

Regularly saving the states of a program, known as checkpointing, is essential to ensure recovery in case of a failure. However, traditional methods of checkpointing can be costly in terms of time and storage space. LC-checkpoint proposes a lossy compression approach to reduce these costs while maintaining an acceptable accuracy of the saved data.

5.2.1.3 Objective

The goal of LC-checkpoint is to minimize the volume of data to be saved and the time required to revert to the last training point before the occurrence of a failure, while preserving the integrity and validity of the data necessary for program resumption.

5.2.1.4 Methodology

LC-checkpoint (see FIGURE 5.1) employs several techniques to efficiently compress data, using a delta encoding scheme, which only tracks the differences between two checkpoints. The algorithm starts by quantifying with exponent-based quantization on the differences between two successive values, then eliminates those updates that are inconsequential with the Priority Promotion technique, and finally compresses further using Huffman encoding. An explanation of what each of these techniques does:

Delta-Encoding Scheme: The delta-encoding scheme is a technique that stores differences (deltas) between successive values rather than the values themselves. This approach is particularly effective when data exhibits some continuity or correlation between successive states. The deltas are generally smaller and contain many zero values, which can be compressed more efficiently than raw values.

Exponent-based Quantization: This technique involves representing values using an exponential base, which allows reducing the precision of values while maintaining an approximate representation sufficient for most applications. By adjusting the exponential base, it is possible to control the level of compression and information loss.

Priority Promotion: Priority Promotion is a technique that prioritizes more critical data during compression. Data with a more significant impact on the accuracy or performance of the program are compressed with less loss, while less critical data can be more aggressively compressed.

Huffman Encoding: Huffman encoding is a lossless compression method that uses variable-length codes to represent data. More frequent data are represented by shorter codes, while less frequent data are represented by longer codes. This reduces the total size of the compressed data.

The compressed data resulting from Huffman encoding are saved, allowing for more efficient resumption of training if it fails for various reasons. This requires the construction of "breakpoints," similar to those used to debug computer programs, so that researchers can easily revert to the state just before the model "crashes" during training. In case of a failure, to revert to the state just before the model "crashes," each saved compressed data is decompressed and then added back to the initial model.

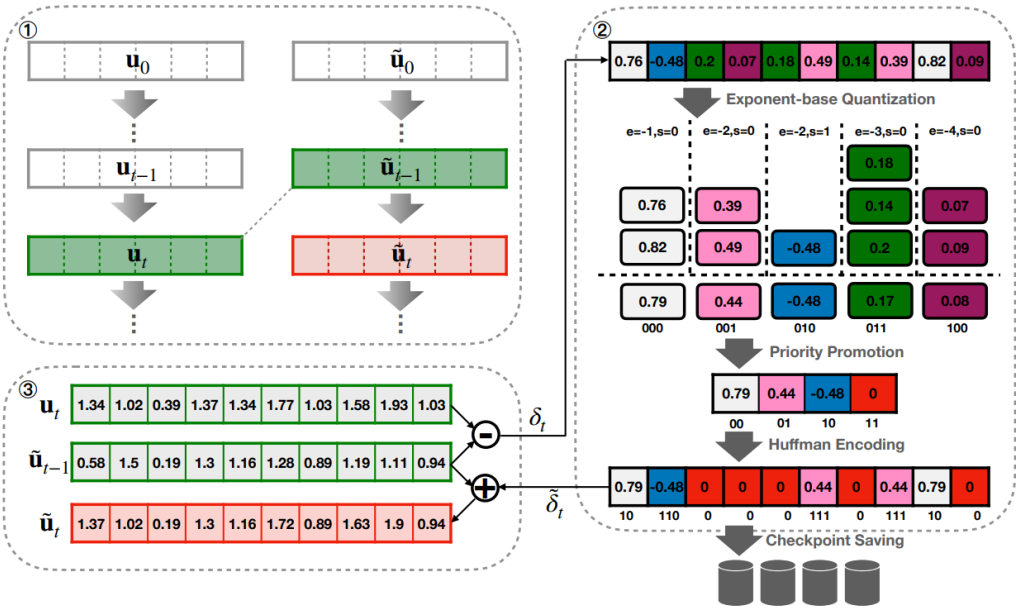

Figure 5.1: Overview of LC-Checkpoint

5.2.1.5 Formulation

More specifically, given a model state u and a local optimal point u^*, the algorithm calculates the update distance δ, quantifies this distance, compresses it using Huffman encoding, and saves it to disk, then updates the checkpoint state. At each step, the system maintains an approximation $\tilde{u}_t$ of the ground truth state. We define simply, where u_0 is the initial state of the model. Our system continuously maintains and updates $\tilde{u}_t$, which is given by:

$$\tilde{u}_t = u_0 + \sum_{i \leq t} \tilde{\delta}_i$$

For quantification, the compression is done in two steps:

- **Exponent-based Quantization**
  - Distribute the entries of δ_t into several groups based on the exponent and identical signs.
  - Represent each group by the average of the maximum and minimum values.
- **Priority Promotion**
  - When $\delta_{t,i} \approx 0$ (i.e., $\tilde{u}_{i,t-1} \approx u_{i,t}$), it is more efficient to group the updates.
  - Preserve the $2^x - 1$ buckets with the largest exponent and merge the remaining buckets into a single one with a value of 0.

5.2.1.6 Utility of LC-Checkpoint

LC-checkpoint is particularly useful for:

- **Storage Size Reduction**: Reduces the amount of disk space required to store checkpoints.
- **Efficiency Improvement**: Reduces the time and resources needed to manage recovery points.
- **Fault Tolerance**: Maintains the ability to efficiently resume even with compressed data.

5.2.1.7 Summary

The LC-checkpoint architecture offers an innovative solution to the checkpointing problem. By combining lossy and lossless compression techniques, such as exponent-based quantization, priority promotion, Huffman encoding, and the delta-encoding scheme, LC-checkpoint significantly reduces storage and time costs while maintaining acceptable accuracy of the saved data. Unlike lossless compression, which retains all original data, lossy compression removes some redundant or less important information, allowing for a significant reduction in the size of data to be stored.

5.2.2 LoRA: Low-Rank Adaptation

5.2.2.1 Introduction

Low-Rank Adaptation (LoRA) [17] is a fine-tuning method, introduced by Edward J. Hu et al. in 2021 to enhance the efficiency of adapting large models, particularly foundation models such as large language models (LLMs). This method allows for the adaptation of models by adding a limited number of parameters while preserving their performance. In other words, LoRA can utilize fewer weights than the total number of weights and is motivated by a 2018 paper discussing the intrinsic dimensionality of large models [18], asserting that there is a low-dimensional parametrization. The idea of LoRA is to assume that, during adaptation, the weight change of the model δW has a low intrinsic dimension to update the weight matrix of the pretrained frozen model W of size $d \times k$:

$$W + \delta W = W + BA \quad \text{avec} \quad B \in R^{d \times r}, A \in R^{r \times k}, r \ll \min(d, k)$$

This section explains the architecture of LoRA, the motivation behind this method, its utility, and how it works.

5.2.2.2 Motivation

With the increasing size of language models, the cost of adapting them (fine-tuning) becomes prohibitive in terms of computational resources and memory. LoRA was developed to address this issue. The main idea is to reduce the number of parameters that need to be adjusted during fine-tuning, which saves resources while maintaining or even improving the model's performance.

5.2.2.3 Architecture of LoRA

The idea of LoRA (see FIGURE 5.2) is to introduce low-rank modifications to the weights of the layers in a pre-trained model, without touching the original weights. More specifically, instead of directly updating the weight matrices W_0 of a layer, LoRA decomposes them into two lower-rank matrices A and B, such that:

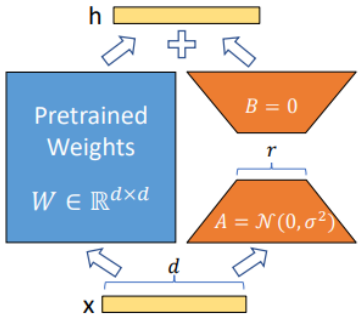

Figure 5.2: Reparametrization with training only A and B

$$W = W_0 + \Delta W \quad \text{avec} \quad \Delta W = A \times B$$

where $A \in R^{d \times r}$ and $B \in R^{r \times k}$, with $r \ll \min(d, k)$, d being the input dimension and k the output dimension of the layer.

Adaptation is only made on matrices A and B, which contain far fewer parameters than W_0. The W_0 parameters remain frozen, which significantly reduces the memory needed and the computational cost of fine-tuning. Furthermore, LoRA can be applied to any dense neural network.

5.2.2.4 Functioning of LoRA

The functioning of LoRA can be summarized in several steps:

1. **Initialization**: The matrices A and B are initialized randomly or by a specific scheme.
2. **Weight Formation**: During the training phase, the modified weights $\Delta W = A \times B$ are added to the pretrained weights W_0 to form the effective weights W of the layer.
3. **Fine-tuning**: Only the matrices A and B are updated during fine-tuning, while W_0 remains unchanged.
4. **Inference**: During inference, the effective weights W are used without requiring significant additional resources.

5.2.2.5 Utility of LoRA

LoRA is particularly useful for:

- **Memory Reduction**: Fewer parameters to store and update, reducing the necessary memory.
- **Computational Efficiency**: Fewer calculations needed during fine-tuning, allowing faster and less resource-intensive adaptation.
- **Flexibility**: The ability to easily transfer and adapt pretrained models to new tasks with minimal resources.

5.2.2.6 Choice of Low Rank

It has been experimentally shown that a rank of 8 is a good balance between compression and performance for large models like LLaMa [19]. Indeed, the observation is that when the rank of the LoRA decomposition for an LLaMa model is 8 or more, the performance of the LoRA does not significantly differ. Although we do not work with LLaMa, we use it as a starting point and define our rank r of the LoRA decomposition of a linear layer $W^{m \times n}$ in our checkpoint mechanism as follows:

$$r = \min(\min(m, n), 8)$$

Further studies could be done to see if other rank values might be feasible.

5.2.2.7 Summary

Low-Rank Adaptation (LoRA) is an innovative method that allows efficient adaptation of large language models while reducing resource costs. This approach is not only economically advantageous but also practical for applications requiring quick and efficient adaptations. The method, proposed by Edward J. Hu and his colleagues in 2021, addresses the crucial problem of scalability in the fine-tuning of large models.

5.2.3 Delta-LoRA: Delta Low-Rank Adaptation

5.2.3.1 Introduction

Delta-LoRA [20] is a modified variant of LoRA (Low-Rank Adaptation), proposed by Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, and Lei Zhang in their paper "Delta-LoRA: Fine-Tuning High-Rank Parameters with the Delta of Low-Rank Matrices." This method aims to optimize high-dimensional parameters using the difference of low-rank matrices. The parameter update is performed as follows:

$$W^{(t+1)} = W^{(t)} + \Delta AB \quad \text{with} \quad \Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$$

Delta-LoRA expands the idea of not only updating the low-rank matrices A and B but also propagating the adaptation to the pretrained weights W through updates using the difference of the product of two consecutive low-rank matrices $(A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)})$.

5.2.3.2 Motivation

The primary motivation behind Delta-LoRA is to solve issues of overfitting and computational efficiency in large neural networks. Traditional fine-tuning methods often require a significant amount of memory and computational power. By using low-rank matrices and focusing on the differences between updates, Delta-LoRA reduces resource needs while maintaining or improving model performance.

Although LoRA significantly reduced the cost of fine-tuning large models, there remain cases where low-rank parameters are insufficient to capture all the complexity of necessary adaptations. Delta-LoRA was developed to address this issue by allowing more fine and precise adaptation of models while retaining the benefits of reduced resource costs.

5.2.3.3 Architecture of Delta-LoRA

The architecture of Delta-LoRA is based on the idea of incorporating differences in products of low-rank matrices to update high-dimensional parameters. Specifically, Delta-LoRA introduces two low-rank matrices A and B for each network layer and updates the layer parameters using the difference between the products of the matrices at two consecutive iterations:

$$\Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$$

This approach allows for more fine and controlled parameter updates, avoiding abrupt jumps that can lead to divergence during learning.

The main idea of Delta-LoRA is to introduce high-rank modifications using the differences (deltas) of low-rank matrices. Instead of simply adjusting the low-rank matrices as in LoRA, Delta-LoRA also refines the differences between these matrices and their initial state.

In practice, Delta-LoRA adds a correction term and updates W as follows:

$$W^{(t+1)} = W^{(t)} + \Delta AB \quad \text{with} \quad \Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$$

Unlike LoRA, in Delta-LoRA, the pretrained weight matrix W_0 can also vary and be updated during the fine-tuning process (see FIGURE 5.3).

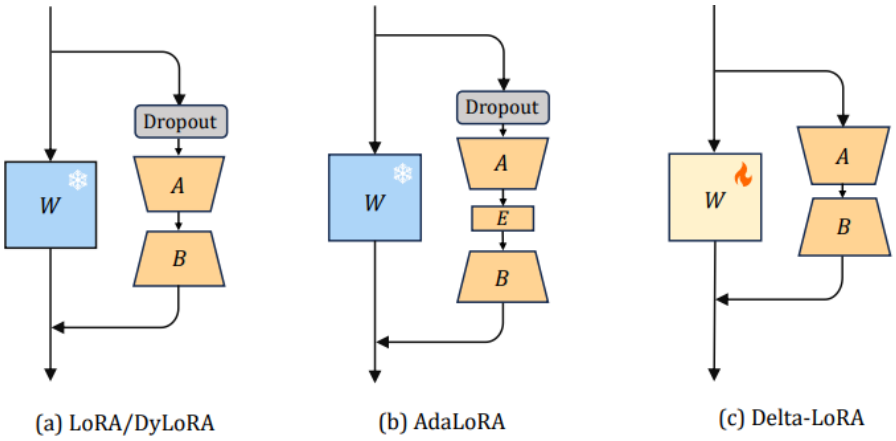

Figure 5.3: Difference between Delta-LoRA and LoRA

5.2.3.4 Functioning of Delta-LoRA

The operation of Delta-LoRA can be summarized in several key steps:

1. Initialization of low-rank matrices A and B for each network layer.
2. At each iteration, compute the products $A^{(t)}B^{(t)}$ and $A^{(t+1)}B^{(t+1)}$.

3. Calculate the difference $\Delta AB = A^{(t+1)}B^{(t+1)} - A^{(t)}B^{(t)}$.
4. Update the layer parameters with $W^{(t+1)} = W^{(t)} + \Delta AB$.

This process allows capturing subtle and relevant changes in parameters, thereby improving model convergence and performance.

5.2.3.5 Use of Delta-LoRA

Delta-LoRA can be used in various contexts for fine-tuning large models, especially in fields like natural language processing, computer vision, and speech recognition. Thanks to its ability to reduce memory and computational needs, this method is particularly suited to resource-limited environments while maintaining high performance.

Using Delta-LoRA is particularly beneficial for:

- **Improving accuracy**: Allows capturing finer variations in parameters, thus enhancing model performance on specific tasks.
- **Memory efficiency**: Although more complex than LoRA, Delta-LoRA remains more memory-efficient than full model fine-tuning.
- **Flexibility**: Facilitates adapting pretrained models to new tasks requiring more precise adjustments.

5.2.3.6 Summary

Delta-LoRA proposes an innovative approach for updating parameters of large models using the differences of products of low-rank matrices. This method combines the advantages of Low-Rank Adaptation (LoRA) with the capability to capture finer variations in model parameters through the deltas of low-rank matrices, offering an effective and high-performance solution for fine-tuning complex neural networks. Delta-LoRA improves the efficiency and accuracy of adapting large models while reducing resource costs.

5.3 Proposed Efficient Checkpointing Scheme

The proposal of this internship is to integrate the respective strengths of LC-checkpoint and Delta-LoRA into a single optimized compression method. This hybrid approach aims to maximize the benefits of each technique, thereby offering a more robust solution for managing checkpoints in model training (see FIGURE 5.4).

The motivation for this scheme is to have even more compressed recovery points than by compressing solely with LC-checkpoint or Delta-LoRA. Like LC-checkpoint, this scheme applies to the pretrained model to which Delta-LoRA layers are added, using various techniques employed in LC-checkpoint, such as calculating consecutive values using the delta-encoding scheme, applying exponent-based quantification to

each delta, then applying priority promotion to emphasize more significant information, and finally applying Huffman encoding.

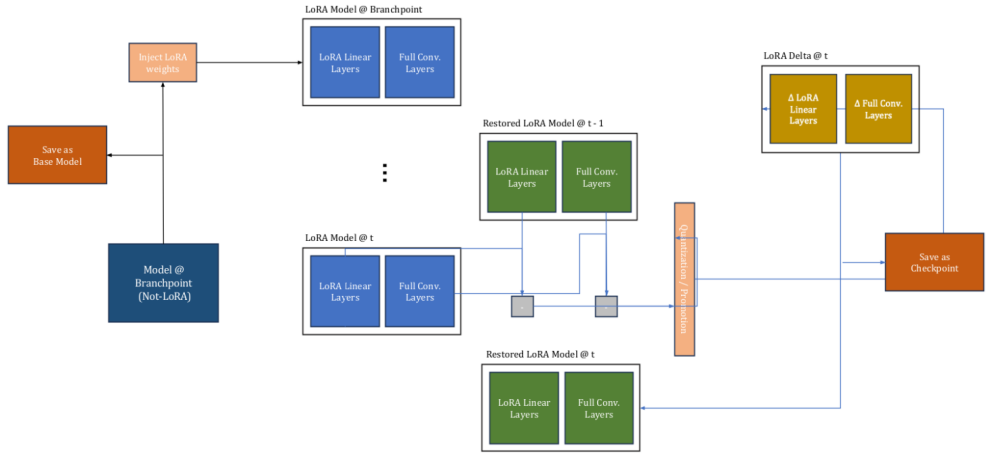

Figure 5.4: Proposed checkpointing scheme

To be more precise, instead of Huffman encoding, the idea is to use GZip, which is a lossless compression method and also directly implementable in Python, the programming language used for this topic.

GZip is a method that combines Huffman encoding [21], LZ77 [22] (another lossless compression method, named after Lempel and Ziv in 1977, which analyzes the input data and determines how to reduce the size of these data by replacing redundant information with metadata), and run-length encoding (RLE), which is also a data compression algorithm that involves replacing sequences of identical values with a pair consisting of the count of repetitions and the value to be repeated.

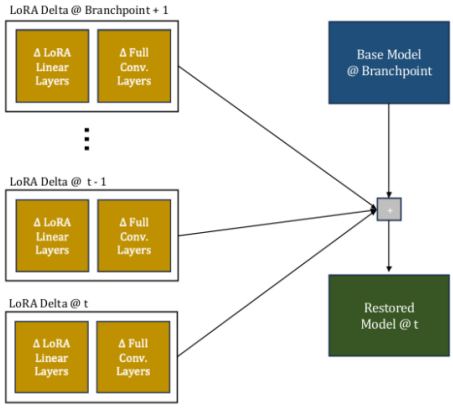

Figure 5.5: Proposed restoration scheme

In the scheme 5.4, you can see in yellow the model backup where dense layers are replaced by Delta-LoRA layers, but also the backup of each checkpoint after compression during adaptation.

For restoration in case of failure (see FIGURE 5.5), the model at the last branch-point is taken and then the value after decompression of the compressed checkpoints up to the last one preceding the failure is added.

Furthermore, to save on computation time for restoration, the model will be updated at periodic iterations, which will be called **supersteps**. Indeed, at every superstep, the model with Delta-LoRA layers is saved (this will be called a **full snapshot**). In this way, in case of failure, the model at the last superstep preceding the failure is taken, and if there are any checkpoints (differences) after this superstep, they are added up to the last checkpoint preceding the failure (see FIGURE 5.6).

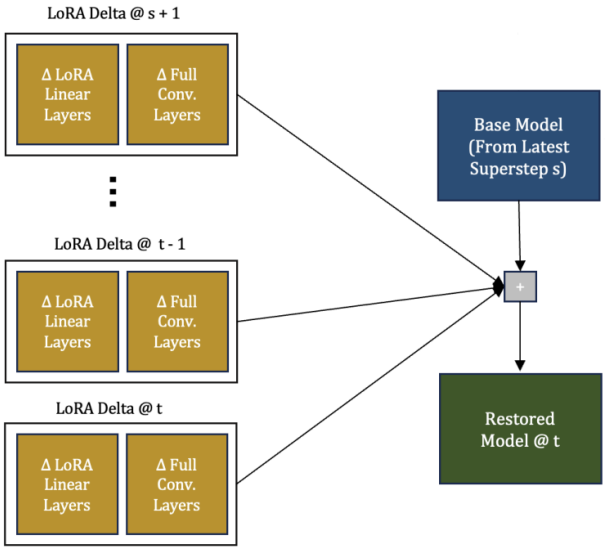

Figure 5.6: Proposed restoration scheme considering the supersteps

6 Details of the Results Obtained

Preliminary results on MNIST using LeNet, AlexNet, and VGG-16 Lite (a version with a single color channel instead of three, whereas VGG-16 was designed to be trained on ImageNet which contains color images) have shown performances comparable to the original finely tuned models, as well as substantial compression of the checkpoints. Similarly, work was carried out on CIFAR-10 with larger and more complex models such as VGG-16 Full, Vision Transformer, and ResNets.

To see the benefit of this compression scheme, a former NUS student demonstrated its value by working on a training set comprising the first 1000 images of the MNIST training set, then validating on the second 1000 images of the MNIST training set, achieving the following results:

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
AlexNet	LC	808.35%	87.629%	98.4% / 98.7% (-0.03%)
	LC + dLoRA	25995.409%	99.615%	95.3% / 98.7% (-3.4%)
VGG-16	LC	813.74%	87.711%	99.1% / 99.4% (-0.03%)
	LC + dLoRA	4188.412%	97.612%	98.4% / 99.4% (-1.0%)
LeNet	LC	537.584%	81.39%	95.9% / 95.9% (-0.0%)
	LC + dLoRA	1889.2869%	94.707%	93.8% / 95.9% (-2.1%)

Table 1: Compression Results

The results were obtained under the following conditions:

Model	AlexNet	VGG-16	LeNet-5
Branching Point	80.72%	72.85%	77.75%
Dataset	MNIST	MNIST	MNIST
Bit-width	3	3	3
LoRA Scaling	0.5	0.5	0.5
Batch Size	32	32	32
Learning Rate	0.01	0.01	0.01
Epochs	20	20	20
Super-Step	Every 10 iteration	Every 10 iteration	Every 10 iteration
Optimizer	SGD[1]	SGD[1]	SGD[1]

Table 2: Test Parameters

To have a pretrained model, for the LeNet-5, AlexNet, and VGG-16 Lite models, the model in question was trained on the entire training dataset without considering the A and B matrices of Delta-LoRA until an accuracy of around 70%-80% was achieved on the entire test set. Then, this pretrained model was used for adaptation (fine-tuning) on the dataset of interest.

6.1 Reproduction of Results

Given that the NUS student had started this work and obtained initial results, I first needed to reproduce his results. However, since some essential files for running the code were missing when this student handed over his code to me, I first had

[1] ¶ SGD or Stochastic Gradient Descent, invented by Robbins H. and Monro S. [23]

to reimplement the missing parts. Then, after reimplementing the missing parts, I obtained the following results for the same models used, adapted (fine-tuned) on the first 1000 images of the MNIST training set and validated on the next 1000 images of the MNIST training set:

Model	**Mechanism**	**Compression Ratio**	**Space Savings**	**Final Accuracy (Restored / Full)**
AlexNet	LC	813.32%	87.705%	98.2% / 99.0% (-0.8%)
	LC + dLoRA	28865.393%	99.654%	96.2% / 99.0% (-3.8%)
VGG-16	LC	813.705%	87.711%	99.1% / 99.4% (-0.3%)
	LC + dLoRA	4185.214%	97.611%	98.1% / 99.4% (-1.3%)
LeNet	LC	562.256%	82.215%	96.1% / 96.1% (-0.0%)
	LC + dLoRA	1885.7610%	94.697%	91.2% / 96.1% (-4.9%)

Table 3: Compression Results for the Reimplementation

The results were obtained under the following conditions:

Model	**AlexNet**	**VGG-16**	**LeNet-5**
Branching Point	80.5%	72.85%	76.8%
Dataset	MNIST	MNIST	MNIST
Bit-width	3	3	3
LoRA Scaling	0.5	0.5	0.5
Batch Size	32	32	32
Learning Rate	0.01	0.01	0.01
Epochs	20	20	20
Super-Step	Every 10 iteration	Every 10 iteration	Every 10 iteration
Optimizer	SGD	SGD	SGD

Table 4: Test Parameters for the Reimplementation

Thus, we can achieve similar compression ratio and final accuracy. Therefore, we keep the code architecture in order to generalize on other models.

6.2 Extending Results to CIFAR-10

After obtaining the compression results on MNIST, I evaluated the compression rate as well as the accuracy on CIFAR-10 using the same models:

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
AlexNet	LC	634.431%	84.238%	99.6% / 99.6% (-0.0%)
	LC + dLoRA	23491.97%	99.574%	94.5% / 99.6% (-5.1%)
VGG-16	LC	814.574%	87.724%	100.0% / 100.0% (-0.0%)
	LC + dLoRA	4756.479%	97.898%	98.2% / 100.0% (-1.8%)
LeNet	LC	415.728%	81.061%	98.4% / 98.4% (-0.0%)
	LC + dLoRA	1698.463%	94.112%	86.0% / 98.4% (-12.4%)

Table 5: Compression Results on CIFAR-10[∓]

The results were obtained under the following conditions:

Model	AlexNet	VGG-16	LeNet-5
Branching Point	80.0%	72.1%	72.0%
Dataset	CIFAR10	CIFAR10	CIFAR10
Bit-width	3	3	3
LoRA Scaling	0.5	0.5	0.5
Batch Size	32	32	32
Learning Rate	0.01	0.1	0.01
Epochs	20	20	20
Super-Step	Every 10 iteration	Every 10 iteration	Every 10 iteration
Optimizer	SGD	SGD	SGD

Table 6: Test Parameters on CIFAR-10

6.3 Extending Results with Other Models: VGG-16 Full, ResNet-50, and Vision Transformer

The objective of this subsection is to assess the portability of our compression approach to more complex and recent neural network architectures, to determine if the compression gains observed with the initial models could be replicated in varied contexts. Thus, we chose to test three popular and structurally different models: VGG-16 Full Version, ResNet-50, and Vision Transformer (ViT).

6.3.1 Testing with VGG-16 Full, ResNet-50, and ViT-Tiny

For optimization of the available time, we maintained the same configuration for both training and evaluation of the models. Training was carried out on the first 1000 images of the MNIST database, and evaluation was performed on the next 1000 images. This uniform method ensures that the results are comparable across different models and configurations.

[1]‡ cf. **Remark 6.3.2**.

[2]∓ cf. section 10.1 for more details about the implementation.

The following results were obtained for the models VGG-16 Full, ResNet-50, and Vision Transformer (ViT).

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
VGG-16†	LC	771.56%	87.04%	99.5% / 99.5% (-0.0%)
	LC + dLoRA	6160.776%	98.378%	98.9% / 99.5% (-0.6%)
ResNet-50	LC	760.59%	86.85%	100.0% / 100.0% (-0.0%)‡
	LC + dLoRA	761.939%	86.876%	99.3% / 100.0% (-0.7%)‡
ViT-Tiny	LC	244.15%	59.04%	76.0% / 76.0% (-0.0%)
	LC + dLoRA	223.928%	55.343%	75.6% / 76.0% (-0.4%)
AlexNet	LC	813.32%	87.71%	98.2% / 99.0% (-0.8%)
	LC + dLoRA	28865.393%	99.654%	96.2% / 99.0% (-2.8%)
VGG-16⋆	LC	811.67%	87.68%	99.9% / 99.9% (-0.0%)
	LC + dLoRA	4562.891%	97.808%	99.1% / 99.9% (-0.8%)
LeNet	LC	562.26%	82.22%	96.1% / 96.1% (-0.0%)
	LC + dLoRA	1885.761%	94.697%	91.2% / 96.1% (-4.9%)

Table 7: Compression Results on MNIST with VGG-16 Full, ResNet-50, and ViT-T∓

The VGG-16 Full Version model, with the application of the LC + dLoRA compression technique, shows impressive compression with minimal degradation in accuracy, suggesting that this compression method is highly effective even for deep and complex network architectures. The ResNet-50, known for its ability to leverage residuals to improve learning, also demonstrated favorable compression rates with minimal losses in accuracy. Finally, the Vision Transformer, a model based on attention mechanisms rather than convolution, achieved significant compression rates with acceptable reductions in accuracy. These experiments confirm the robustness of our compression approach across different model architecture paradigms and highlight the importance of wisely choosing compression configurations to maintain a balance between efficiency and accuracy.

It is noteworthy that although ResNet-50 shows a significant compression rate, this rate remains relatively modest compared to older models like VGG-16. This observation can be explained by the architectural structure of ResNet-50 itself. Unlike VGG-16, which contains several fully connected layers that are generally rich in parameters, ResNet-50 uses a reduced number of such layers. Indeed, the majority of the ResNet-50 architecture is composed of residual blocks that contain only a

[1] † VGG-16 Full Version
[2] ⋆ VGG-16 Lite Version
[3] ‡ cf **Remark 6.3.2**.
[4] ∓ cf. section 10.1 for more information about the implementation.

small number of fully connected layers. Consequently, the application of compression techniques like Delta-LoRA, which primarily target these layers, does not result in as substantial a reduction in the total number of parameters. This architectural specificity underscores the importance of adapting compression strategies to the peculiarities of each model to maximize efficiency without compromising the accuracy of predictions.

Regarding the Vision Transformer - Tiny (ViT-Tiny) mentioned in the table, this lightweight version was chosen to begin our experiments due to computational considerations, allowing us to test our compression methods on architectures based on attention mechanisms while limiting the required resources. The ViT-Tiny is configured with a number of attention heads ('n _heads') equal to 2, a number of blocks ('n _blocks') of 2, a hidden dimension ('hidden _dim') of 8, and a dimension for the multi-layer perceptron layers ('mlp _size') of 24. Given that the linear layers in the multi-head self-attention blocks do not contain as many parameters as the fully connected layers of more traditional architectures, the compression gains on these layers are also limited. This once again illustrates the importance of adapting compression techniques to the structural specifics of each model to optimize both compression and performance.

6.3.2 Vision Transformer-Small and Study of Delta-LoRA Application Layers

Due to the insufficient compression obtained with ViT-Tiny on MNIST, we decided to develop a ViT-Small (ViT-S) model, which is a more advanced configuration and closer to the smallest ViT model available in the original paper. ViT-S is equipped with eight attention heads ('n _heads'), eight blocks ('n _blocks'), a hidden dimension ('hidden _dim') of 512, and a multi-layer perceptron layer dimension ('mlp _size') of 2048. This more robust configuration aims to explore the compression potential on a model that incorporates more complex structural features. Additionally, as part of our experiments with ViT-Small, we evaluated the application of the Delta-LoRA technique not only on the fully connected layers but also specifically on the components of the multi-head self-attention (MHSA or MSA), particularly on the 'query' and 'value' layers as suggested in the LoRA research paper. This approach aimed to determine which layers benefit most from Delta-LoRA compression, leading to notable results documented below.

Model	Mechanism	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
ViT-S (MSAxMLP)	LC	764.484%	86.919%	97.3% / 97.3% (-0.0%)
	LC + dLoRA	12644.316%	99.209%	91.4% / 97.3% (-5.9%)
ViT-S (MSA)	LC	767.055%	86.963%	96.7% / 96.7% (-0.0%)
	LC + dLoRA	790.168%	87.344%	91.4% / 96.7% (-5.3%)
ViT-S (MLP)	LC	761.956%	86.876%	97.0% / 97.2% (-0.2%)
	LC + dLoRA	8215.592%	98.783%	96.7% / 97.2% (-0.5%)

Table 8: Compression Ratio Comparison on MNIST between ViT-S (MSAxMLP), ViT-S (MSA), and ViT-S (MLP)

We observe that the compression rate is significantly higher when the Delta-LoRA approach is applied to MSAxMLP compared to its application on MSA and on MLP. This is intuitive, as adding Delta-LoRA to MSAxMLP compresses a larger number of parameters, thus increasing the efficiency of the compression. The compression rate of Delta-LoRA on MSA and MLP lies between that of the application of Delta-LoRA on MSA and that on MSAxMLP. This rate is closer to that obtained with MSAxMLP, which makes sense since the number of parameters in MLP is more significant than in MSA. In conclusion, Delta-LoRA demonstrates an additive effect: the more parameters there are to compress, the higher the compression rate.

Remark: *The results obtained, particularly the 100% accuracy for ResNet-50 on MNIST and for the lightweight version of VGG-16 on CIFAR-10, suggest a situation of overfitting. Upon verification, it was confirmed that our test of overfitting was accurate: the images in the training set and the evaluation set were highly similar. Therefore, the values obtained should be interpreted with caution. This awareness came somewhat late in the process. In the continuation of the internship, we took this issue into account by focusing on the implementation of incremental learning. The development of this new approach is detailed in the following section, where we work on the complete set of data of interest. It is important to note that the compression rate is independent of accuracy, meaning these overfitting issues do not affect our compression measurements.*

7 Data Poisoning and Incremental Learning

7.1 Motivation Behind the Application of Incremental Learning

The implementation of incremental learning in our checkpointing scheme is motivated by the need to address the problem of *learning-to-defer*, where the goal is to enable the AI to delegate decisions to human experts, whether malicious or not,

when its accuracy is insufficient. This strategy is essential to avoid hallucination issues, which are common in AI systems. When a delegation is made, the AI learns from the expertise provided by the human. Therefore, it is crucial to train the model to anticipate potential malicious behavior. Thus, if such behavior is detected, training can be resumed just before this detection.

In the traditional training context of deep neural networks, it is generally assumed that the dataset is correct by default. The model learns by going through this set at each epoch and repeats this process several times until a stopping condition is reached: either validation accuracy stagnation, insufficient error reduction, or the predefined number of epochs is reached. However, incremental learning proposes a new approach in the field of deep learning. This learning mode allows the model to continuously learn from incoming data, adapting to each new dataset encountered. In our project, we initially focus on scenarios where each new dataset comes from the same original source (such as MNIST or CIFAR-10) and contains all classes from the original dataset.

By integrating data poisoning into our checkpointing scheme, we address a crucial aspect of machine learning model security and robustness. This mechanism aims to ensure that the model remains reliable and performant, even in the presence of potentially corrupted data, thereby preserving the integrity of learning in the face of potential attacks.

7.2 Assumptions on the Dataset Partitioning

For the specific partitioning of the dataset intended for training, we adopt a strategy that ensures adequate diversity and representativeness within each subset while maintaining the integrity of the evaluation set, which remains unchanged.

The partitioning is structured as follows:

- **Four dataloaders** each containing **25%** of the dataset of interest. Each dataloader is designed to ensure that the images are different from one another and that all classes of the dataset of interest are represented. This configuration allows for deep and robust learning from substantial segments of the dataset.
- **Twenty dataloaders** each containing **5%** of the dataset of interest. They ensure that the images maintain a complete representation of the classes, thereby increasing the frequency of model updates and the diversity of data seen by the model during an epoch.
- **Eighty dataloaders** each containing **1.25%** of the dataset of interest. These small dataloaders offer even finer granularity, ideal for quick tests and frequent model adjustments, allowing for precise and targeted fine-tuning.

A major challenge in incremental learning is the phenomenon of *catastrophic forgetting*, which describes the tendency of neural network models to forget previously learned information when exposed to new data. This concept is intuitive and natural, similar to the human brain, which can forget previously learned information when processing new information. In the context of incremental learning, each new dataloader can potentially erode the knowledge acquired from previous dataloaders [26].

This meticulous structuring is essential to support our incremental learning objectives, where each new portion of data should enrich the model without reintroducing previously corrected biases.

7.3 Results considering Incremental Learning

7.3.1 Study of training set split configurations of interest

For our study, we considered the three previously mentioned splits for incremental learning, and the following results were obtained for the LeNet-5 model on MNIST (with the configurations to be provided after the results):

Split	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
4*25% (15k-15k-15k-15k)	LC	888.872%	88.75%	98.87%/98.85% (+0.02%)
	LC + dLoRA	2481.857%	99.209%	92.09%/98.85% (-6.76%)
20*5% (20* 3k)	LC	723.688%	86.182%	98.39%/98.37% (+0.02%)
	LC + dLoRA	2270.412%	95.596%	92.63%/98.37% (-5.74%)
80* 1.25% (80* 750)	LC	885.524%	88.71%	97.65%/97.75% (-0.1%)
	LC + dLoRA	2478.693%	95.97%	94.08%/97.75% (-3,67%)

Table 9: Comparison of the compression rate on MNIST for LeNet-5 across the three configurations: 4*25%, 20*5%, and 80*1.25%

The compression rate does not differ significantly depending on the configurations. However, in the 80*1.25% configuration, the performance loss is less significant than in the others. The model, therefore, learns better in incremental learning when there is less data to learn.

Next, a study was also conducted for the 4*25% case on ViT-B/16, and the following results were obtained:

Split	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
4*25% (4* 12.5k)	LC	591.368%	83.083%	98.70%/98.74% (-0.04%)
	LC + dLoRA	8173.368%	98.777%	97.79%/98.74% (-0.95%)

Table 10: Compression ratio on CIFAR-10 with ViT-B/16 for the 4*25% configuration

All the previous results were obtained using the parameters aggregated in the table below, where the same parameters and hyperparameters were used to ensure a fair comparison and to obtain comparable results.

Dataset Splitting	4*15k (=25%)	20*3k (=5%)	80*750 (=1.25%)	4*12.5k (=25%)
Model	LeNet-5			ViT-B/16
Branching Point	74.6%			HF[‡] pretrained ImageNet1k
Dataset	MNIST			CIFAR-10
Bit-width	3			3
LoRA Scaling	0.5			0.5
Batch Size	128			128
Learning Rate	0.08			4e-5
Epochs	100			5
Super-Step	Every 10 iterations			Every 10 iterations
Stopping Criterion	Early stopping			Early stopping
Optimizer	SGD			Adam[¶]

Table 11: Comparison of LeNet-5 and ViT-B/16 configurations across different datasets

For more information on the evolution of performance throughout the dataloaders during incremental learning, the following tables aggregate this information:

[1]‡ HF = Hugging Face. The branching point was retrieved from the Hugging Face website

[2]¶ Adam or Adaptive Moment Estimation [?] is a variant of SGD, popular for its ability to adapt to the specific problem by adjusting the learning rates

Split	Methods	Compression ratio	Space Savings	Full Validation Accuracy (Restored / Full)
4* 25% (15k-15k-15k-15k)	LC + dLoRA	2481.857%	95.97%	88.74%-90.22%-91.33%-**92.09%/98.85% (-6.76%)**
	LC	888.872%	88.75%	98.11%-98.70%-98.83%-**98.87%/98.85% (+0.02%)**
	nothing	100.00%	0%	98.13%-98.73%-98.83%-98.85%
20* 5% (20* 3k)	LC + dLoRA	2270.412%	95.596%	84.36%-[...]-**92.63%/98.37% (-5.74%)**
	LC	723.688%	86.182%	93.81%-[...]-**98.39%/98.37% (+0.02%)**
	nothing	100.00%	0%	93.82%-[...]-98.37%
80* 1.25% (80* 750)	LC + dLoRA	2478.693%	95.97%	81.16%-[...]-**94.08%/97.75% (-3,67%)**
	LC	885.524%	88.71%	87.95%-[...]-**97.65%/97.75% (-0.1%)**
	nothing	100.00%	0%	88.00%-[...]-97.75%

Figure 7.1: Performance evolution for LeNet-5 during incremental learning for the different configurations

4* 25% (4*12.5k)	LC + dLoRA	8173.368%	98.777%	**97.79% / 98.74% (-0.95%)**
	LC	591.368%	83.083%	**98.70% / 98.74% (-0.04%)**
	nothing	100.00%	0%	98.74%

Figure 7.2: Performance evolution for ViT-B/16 during incremental learning for the different configurations

7.3.2 Rank analysis in Incremental Learning

Since the optimal rank can change from one dataloader to another, a study of rank was conducted. An initial study was carried out using constant ranks, meaning the same rank for each dataloader, and preliminary results were obtained for Vision Transformer-B/16 pre-trained on ImageNet-1k, adapted to the CIFAR-10 training set:

Rank	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
16	LC	534.542%	81.292%	98.27%/98.23% (+0.04%)
	LC + dLoRA	7325.409%	98.63%	96.56%/98.23% (-1.67%)
4	LC	534.414%	81.29%	98.27%/98.21% (+0.06%)
	LC + dLoRA	8675.518%	98.847%	96.71%/98.21% (-1.5%)

Table 12: Comparison of compression rates for different ranks in the 4*25% configuration with rank = 16 and rank = 4

Results obtained under the following conditions:

Rank	16	4
Delta-LoRA Layers application	MSA x MLP of the Transformer Encoder	
Model	ViT-B/16	
Branching Point	Hugging Face (pre trained on imagenet-1k)	
Dataset	CIFAR-10	
Dataset splitting	25%-25%-25%-25%	
Bit-width	3	
LoRA Scaling	0.5	
Batch Size	128	
Learning Rate	4e-5	
Epochs	1	
Super-Step	Every 10 iterations	
Stopping Criterion	Early stopping	

Table 13: Configuration for the comparison of compression rates for rank = 16 and 4 in the 4*25% configuration

Next, observing that Vision Transformer-B/16 pre-trained on ImageNet-1k achieves similar performance for different ranks but yields more interesting compression rates with a lower rank, a second study was conducted to further explore this lower rank in the same context, while also using more epochs, resulting in:

Rank	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
4	LC	554.215%	81.956%	98.47%/98.5% (-0.03%)
	LC + dLoRA	8675.55%	98.847%	97.53%/98.5% (-0.91%)
3	LC	959.664%	89.58%	98.51%/98.51% (-0.00%)
	LC + dLoRA	8810.807%	98.865%	97.53%/98.51% (-0.92%)
2	LC	553.623%	81.937%	98.49%/98.49% (-0.00%)
	LC + dLoRA	8950.415%	98.883%	97.48%/98.49% (-1.01%)
1	LC	959.664%	89.58%	98.33%/98.51% (-0.18%)
	LC + dLoRA	9094.627%	98.9%	97.56%/98.51% (-0.95%)

Table 14: Comparison of compression rates for different ranks in the 4*25% configuration for ViT-B/16 pre-trained on ImageNet-1k, adapted to CIFAR-10

These results show that the performance is similar for different ranks of 1, 2, 3, and 4. The only difference lies in the compression rate, where a rank of 1 achieves the best compression rate, which is intuitive since a rank of 1 indicates less information and therefore more compression. Thus, a rank of 1 is sufficient to maintain performance as good as the others on CIFAR-10, possibly due to the simplicity of the dataset's classification task. Given the previous result, the task needs to be more complex by using a more challenging dataset. Therefore, the Stanford Cars Dataset [30] was suggested, containing 16,185 images and 196 classes. Additionally, since we also want to observe the relevance of our scheme with larger models, we decided to perform the classification with a ViT-L/16 pre-trained on ImageNet-21k:

Rank	Methods	Compression Ratio	Space Savings	Final Accuracy (Restored / Full)
8	LC	766.253%	86.949%	76.81%/77.19% (-0.38%)
	LC + dLoRA	7036.127%	98.579%	10.53%/77.19% (-66.66%)

Table 15: Compression rate and performance in the 4*25% configuration for ViT-L/16 pre-trained on ImageNet-21k, adapted to Stanford Cars

Results obtained under the following conditions:

Rank	8
Delta-LoRA Layers application	MSA x MLP of the Transformer Encoder
Model	ViT-L/16
Branching Point	Hugging Face (pretrained on ImageNet-21k)
Dataset	Stanford Cars
Dataset splitting	25%-25%-25%-25%
Bit-width	3
LoRA Scaling	0.5
Batch Size	64
Learning Rate	5e-5
Learning Rate Delta-LoRA + LC	7e-5
Epochs	3
Super-Step	Every 10 iterations
Optimizer	AdamW

Table 16: Configuration for compression rate with rank = 8 in the 4*25% setup for ViT-L/16 on Stanford Cars

The performance of the model following our Delta-LoRA + LC scheme showed a significant performance drop compared to the model without our scheme. This may be due to several factors, the first being the amount of images per class used for fine-tuning, and the second being the number of epochs necessary for fine-tuning. These results were obtained using the NSCC server, connecting to the ASPIRE 2A supercomputer in Singapore, and the process took 1 day and 16 hours, primarily due to the time required to calculate the performance of each model in our learning loop on the validation set.

For fine-tuning, the hyperparameters as well as the choice of optimizer (AdamW [31]) were finely adjusted to maintain a validation accuracy around 87%, as shown in the evolution of accuracy and loss in FIGURES 7.3 and 7.4.

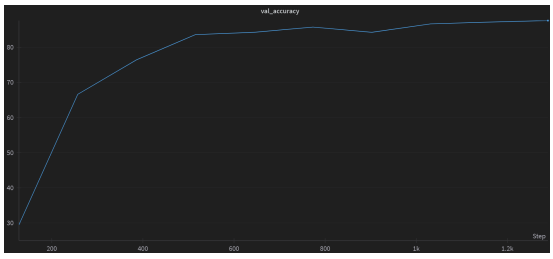

Figure 7.3: Validation accuracy of ViT-L pretrained on ImageNet-21k, adapted to Stanford Cars

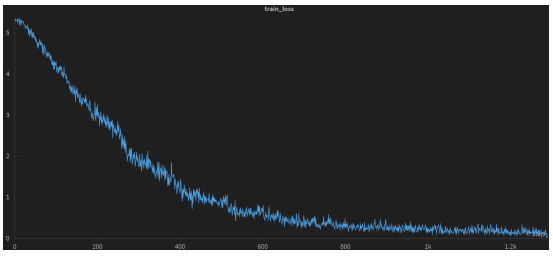

Figure 7.4: Train Loss of ViT-L pretrained on ImageNet-21k, adapted to Stanford Cars

Nevertheless, it should be noted that the configuration using Delta LoRA + LC

required more epochs to reach an accuracy comparable to the configuration without Delta LoRA + LC, which led to a considerable increase in training time that we could not afford due to time constraints. Furthermore, the decision to apply Delta-LoRA to the MSA and MLP layers of the Transformer encoder was motivated by the desire to achieve a significant compression rate. However, the performance drop might have been caused by this excess compression, suggesting that applying it only to the MSA layers of the Transformer encoder could potentially be more appropriate.

In the continuation of this study on incremental learning with variable ranks, we adapted a Vision Transformer-Large model (ViT-L/16) pretrained on ImageNet-21k to classify the Stanford Cars dataset, considering a possible optimization for dataloader splitting. With a training set of 8144 images spread across 196 classes, this gives an average of 42 images per class. Dividing this set into four equal-sized dataloaders would mean each dataloader contains around ten images per class, which could be considered insufficient for effective learning. To address this issue, a second study was conducted where the first dataloader encompasses 70% of the training set (ensuring at least one image per class), and the remaining 30% is equally distributed across the following three dataloaders, 10% each. This configuration could potentially allow the model following our Delta LoRA + LC scheme to benefit from sufficient data for effective learning, achieve a good initial accuracy, and continue to improve its performance with each new dataloader.

This approach aims to maximize the benefits of incremental learning while adjusting the granularity of processed data to optimize the model's performance throughout the fine-tuning process. However, due to time constraints, this study could not yield conclusive results, and further in-depth research could be conducted.

8 Dataset Verification

In any deep learning model training process, the quality of the data used is crucial to ensure the generalization and robustness of the model. A fundamental aspect is the prior verification of the diversity and representativeness of the images present in the training and test sets.

8.1 Verification of Unique Images

Before proceeding with training, it is essential to ensure that the images in the training set are distinct from those in the test set. This verification prevents the model from simply memorizing the images seen during training, a phenomenon that leads to overfitting, rendering the model incapable of generalizing to new data. A manual inspection of the images can be employed for this task (see FIGURE 8.1), ensuring that each image in the test set is genuinely unknown to the model.

As previously mentioned, overfitting can skew the obtained results 6.3.2, and this awareness motivated adjustments in the methodology employed subsequently.

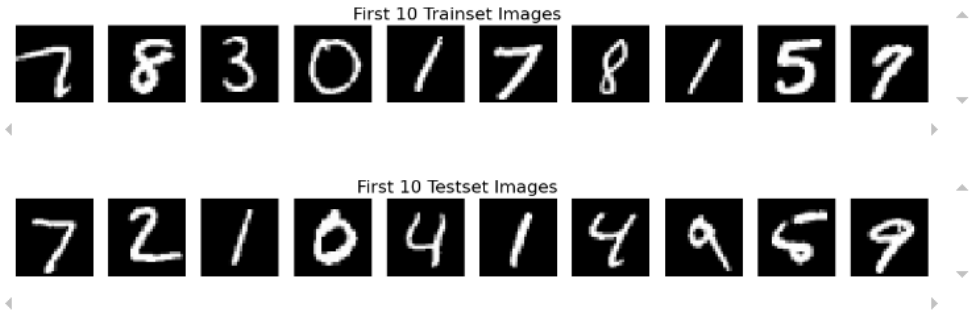

Figure 8.1: Verification of image differences between the training and test sets for MNIST

As illustrated in Figure 8.1, it can be observed that the first ten images are different between the training and test sets. This visual distinction is crucial to prevent overfitting and ensures that the model can effectively generalize to new data. Verifying that the datasets do not contain identical images is an essential step in assessing the model's generalizability, thereby strengthening the robustness of the obtained results.

8.2 Verification of the Presence of All Classes in the Dataloaders

In the context of our incremental learning approach, where multiple segmentations of the training set are considered, the assumption is that each dataloader must be carefully designed to include all classes present in the dataset of interest.

For each dataloader, a test is performed to ensure that all classes are well-represented (see FIGURE 8.2). This verification helps maintain the diversity of data seen by the model at each stage of training, thereby maximizing its generalization capacity.

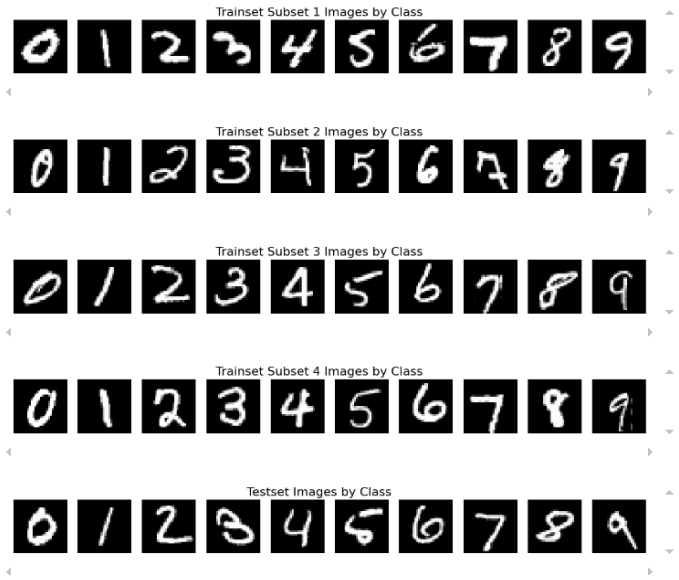

Figure 8.2: Verification of the presence of all classes in each dataloader for MNIST

Regarding the Stanford Cars dataset, an analysis of the presence of classes in the dataloaders was also conducted (see FIGURE 8.3, FIGURE 8.4, FIGURE 8.5, and FIGURE 8.6).

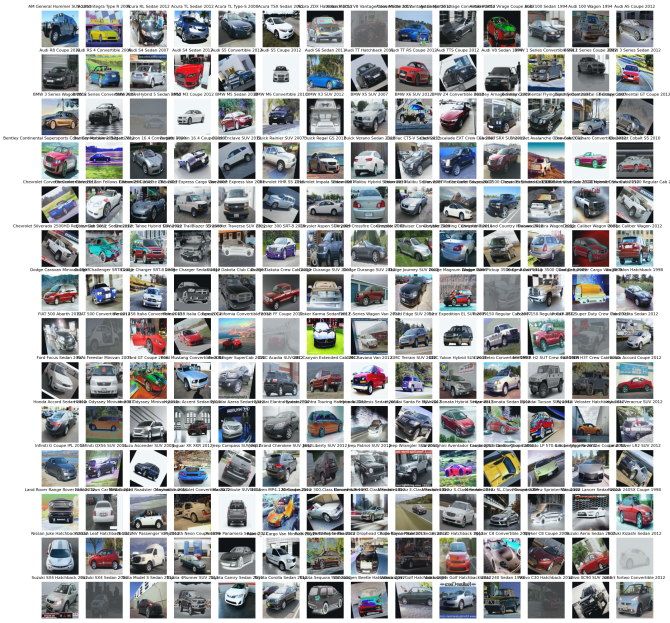

Figure 8.3: Verification of the presence of all classes in Dataloader 1 for Stanford Cars

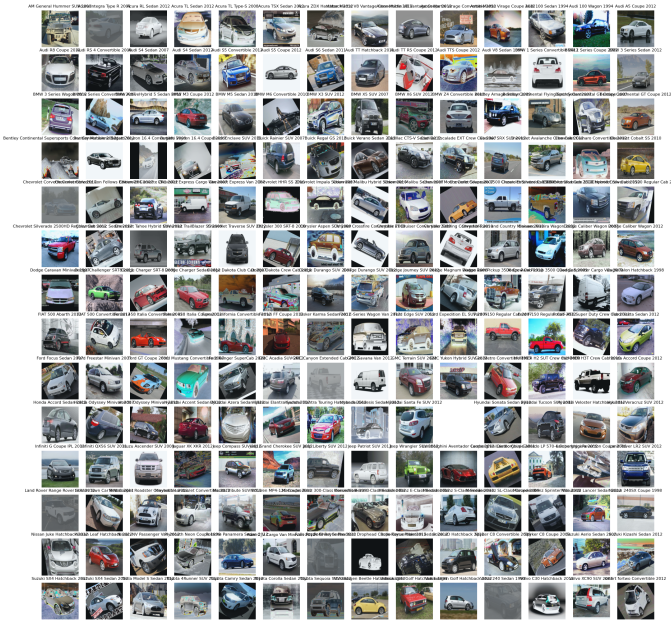

Figure 8.4: Verification of the presence of all classes in Dataloader 2 for Stanford Cars

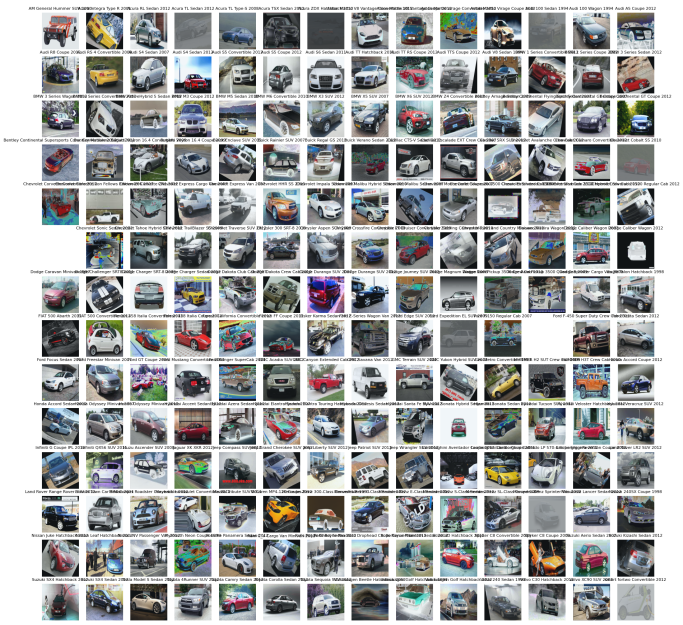

Figure 8.5: Verification of the presence of all classes in Dataloader 3 for Stanford Cars

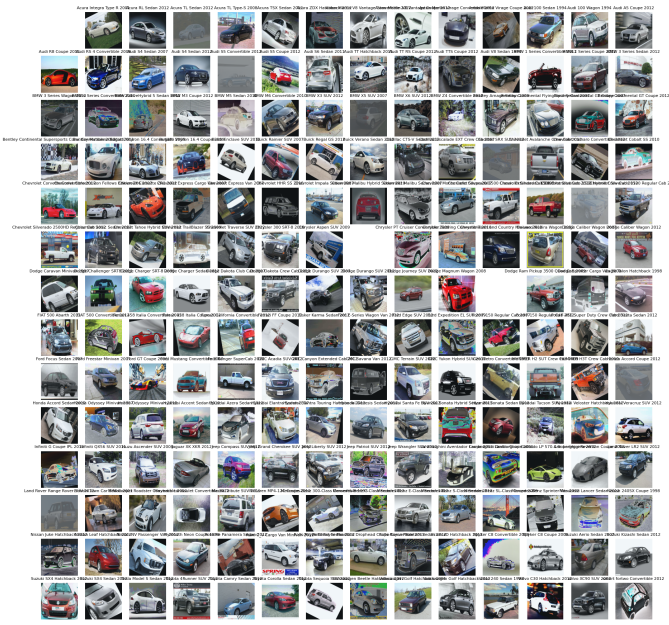

Figure 8.6: Verification of the presence of all classes in Dataloader 4 for Stanford Cars

8.3 Summary

By integrating these preliminary checks, the model is trained under optimal conditions, minimizing the risks of overfitting and maximizing its ability to generalize. These critical steps help to reinforce confidence in the obtained results and ensure the robustness of the model when faced with new data.

9 Explanation of Model Choices

Focusing solely on image classification for this project, the motivation behind the choice of our models was based on both their performance and their significant use in image classification. Additionally, we followed models that adhered to the chronological order of their development, as shown in FIGURE 9.1 [25], where each model addresses the limitations encountered by the previous models.

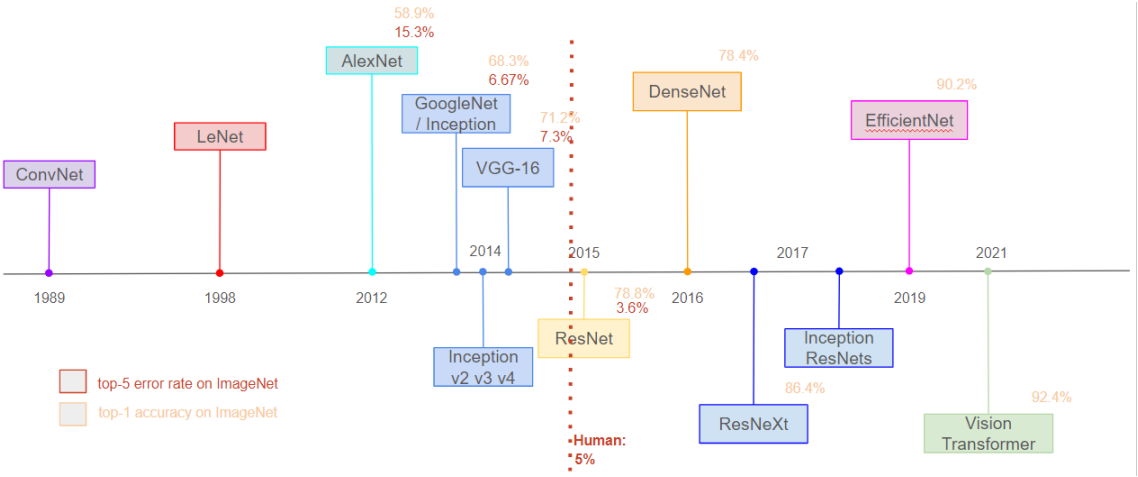

Figure 9.1: Chronology of Image Classification Models

10 Pseudo-code

The codes are available at the following link: https://github.com/BryanBradfo/pfe_lc_lora. Below are the pseudo-codes explaining the implementation of the efficient checkpointing scheme, both without considering incremental learning and then considering data poisoning.

10.1 Efficient Checkpointing: Without Incremental Learning

Algorithm 1 Pseudo-code of the proposed scheme on a ViT

```
Initialize the layers to decompose, the rank for Delta-LoRA decomposition
Initialize the learning rates for the model with and without using the scheme
Load the pre-trained ViT model for image classification
Initialize the optimizers for each model
Initialize iterators and sets for tracking the training
for each batch in the training set do
    Train the models with and without using the proposed scheme
    Save the model states at specified intervals
    Evaluate the accuracy of the models at specified intervals
end for
```

10.2 Efficient Checkpointing: With Incremental Learning

Algorithm 2 Pseudo-code of the proposed scheme on a ViT

```
Initialize the layers to decompose, the rank for Delta-LoRA decomposition
Initialize the learning rates for the model with and without using the scheme
Load the pre-trained ViT model for image classification
Initialize the optimizers for each model
Define lists to record accuracies
Initialize iterators and sets for tracking the training
for each data batch in the training set do
    for each mini-batch in the data batch do
        Train the models with and without using the proposed scheme
        Save the model states at specified intervals
        Evaluate the accuracy of the models at specified intervals
        Apply early stopping based on accuracy improvement
    end for
    Display the results
    Reset configurations for the next data batch
end for
```

11 Conclusion

11.1 Summary of Results and Future Perspectives

The value of an innovative checkpointing scheme has been demonstrated throughout this report, leveraging the potential of LC-checkpoint and Delta-LoRA. Results were obtained for several deep neural network architectures, including LeNet-5, AlexNet, VGG-16 lite version, VGG-16 full version, and Vision Transformer. These experiments were conducted on the CIFAR-10 and MNIST datasets, and there is potential to extend this work to other datasets such as ImageNet-1k or to other domains like Natural Language Understanding (NLU), by exploring models like DistilBert, RoBERTa, and gpt2 on datasets like IMDb.

The choice of dataset segmentation for incremental learning, as well as the rank selection for Delta-LoRA, may be subject to criticism. Future research could explore the adoption of an increasing rank as new data accumulates, potentially mitigating the reduction in accuracy faced with increasing complexity in the knowledge to be integrated.

Although the work conducted is preliminary for a scientific publication, it opens promising avenues for further research. A more thorough reimplementation of the LC-checkpoint and Delta-LoRA techniques is necessary, as their code has not been

previously published, and only a superficial verification has been conducted so far.

Additional efforts could focus on data compression, which would optimize computation time by enabling faster matrix operations in Python, rather than point-by-point calculations. These improvements would significantly increase the efficiency of the methods employed.

Moreover, a potential criticism of this scheme could be the choice of algorithms used. While LC-checkpoint was considered state-of-the-art in 2020, new developments such as QD-Compressor [27] in 2023, DynaQuant [28] in 2023, and ExCP [29] in 2024 have since emerged. Therefore, it would be relevant to consider adapting our scheme by integrating these new checkpoint compression methods. However, incorporating these technologies would require thorough research. Similarly, Delta-LoRA, introduced in 2023, could also benefit from a comparative evaluation with other recent innovations in the field.

11.2 Technical Skills Developed

This internship was particularly transformative from a technical perspective, integrating and deepening many of the knowledge acquired at ENSEEIHT. It introduced me to the exciting world of research in artificial intelligence, exposing me to the rigorous work methods required in this field. During this internship, I explored advanced techniques such as LoRA - Low Rank Adaptation, quantization, as well as LC-checkpoint, which incorporates various data compression methods. I also had the opportunity to implement and experiment with renowned image classification models such as LeNet-5, AlexNet, VGG-16, ResNet-50, and Vision Transformer (ViT), while delving deeply into the self-attention mechanism within the ViT model.

The use of platforms such as Hugging Face facilitated access to and manipulation of open-source models, while tools like FileZilla and PuTTY were essential for managing files and tasks on the ASPIRE 2A supercomputers in Singapore via NSCC. Additionally, I was able to analyze in detail the behavior of models during fine-tuning using visualization tools like Weight and Bias (wandb).

This internship not only enriched my technical understanding and knowledge of cutting-edge AI technologies but also strengthened my ability to work in a demanding research environment, while cultivating rewarding professional relationships. These experiences have confirmed the importance of continuous innovation and collaboration in the field of artificial intelligence research.

References

[1] Dominique Baillargeat, *CNRS@Create*. CNRS - Centre national de la recherche scientifique, 2019.

[2] Heng Swee Keat, *National Research Foundation*. Deputy Prime Minister and Coordinating Minister for Economic Policies, 2006.

[3] Yu Chen, Zhenming Liu, Bin Ren, Xin Jin, *On Efficient Constructions of Checkpoints*. arXiv:2009.13003, 2020. [cs.LG]

[4] Bojia Zi, Xianbiao Qi, Lingzhi Wang, Jianan Wang, Kam-Fai Wong, Lei Zhang, *Delta-LoRA: Fine-Tuning High-Rank Parameters with the Delta of Low-Rank Matrices*. arXiv:2309.02411, 2023. [cs.LG]

[5] Yann LeCun, Leon Bottou, Yoshua Bengio, Patrick Haffner, *Gradient Based Learning Applied to Document Recognition*. Proceedings of the IEEE, vol. 86, no. 11, pp. 2278-2324, 1998.

[6] Alex Krizhevsky, Ilya Sutskever, Geoffrey E. Hinton, *ImageNet Classification with Deep Convolutional Neural Networks*. Advances in Neural Information Processing Systems 25 (NeurIPS 2012), University of Toronto, 2012.

[7] Karen Simonyan, Andrew Zisserman, *Very Deep Convolutional Networks for Large-Scale Image Recognition*. arXiv preprint arXiv:1409.1556, Oxford University, 2014.

[8] Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun, *Deep Residual Learning for Image Recognition*. arXiv:1512.03385, Microsoft, 2015.

[9] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, Neil Houlsby, *An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale*. arXiv:2010.11929, Google, 2020.

[10] Yann LeCun, Corinna Cortes, Christopher J.C. Burges, *Modified National Institute of Standards and Technology database*. 1994.

[11] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, *Canadian Institute For Advanced Research - 10*. 2009.

[12] Alex Krizhevsky, Vinod Nair, Geoffrey Hinton, *Canadian Institute For Advanced Research - 100*. 2009.

[13] Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, Li Fei-Fei, *ImageNet Large Scale Visual Recognition Challenge*. arXiv:1409.0575, Stanford, 2013.

[14] Chen Y., Liu Z., Ren B., Xin J., *On Efficient Constructions of Checkpoints*. arXiv:2009.13003, 2020.

[15] Goodfellow I., Bengio Y., and Courville A., *Deep learning*. MIT press, 2016

[16] Ioffe S. and Szegedy C., *Batch normalization: Accelerating deep network training by reducing internal covariate shift*. arXiv:1502.03167, 2015.

[17] Hu E., Shen Y., Wallis P., Allen-Zhu Z., Li Y., Wang L., Chen W., *LoRA: Low-Rank Adaptation of Large Language Models*. arXiv:2106.09685, 2021.

[18] Li C., Farkhoor H., Liu R., Yosinski J., *Measuring the Intrinsic Dimension of Objective Landscapes*. arXiv:1804.08838, 2018.

[19] Niederfahrenhorst A., Hakhamaneshi K., Ahmad R., *Fine-Tuning LLMs: LoRA or Full-Parameter? An in-depth Analysis with Llama 2*. anyscale, 2023.

[20] Zi B., Qi X., Wang L., Wang J., Wong K., Zhang L., *Delta-LoRA: Fine-Tuning High-Rank Parameters with the Delta of Low-Rank Matrices*. arXiv:2306.10925, 2023.

[21] David A. Huffman, *A Method for the Construction of Minimum-Redundancy Codes*. Proceedings of the IRE, Vol. 40, No.9, pp. 1098-1101, Sept. 1952.

[22] Ziv J., Lempel A., *A Universal Algorithm for Sequential Data Compression*. IEEE Transactions on information theory, Vol. it-23, No.3, May 1977.

[23] Robbins H., Monro S., *A Stochastic Approximation Method*. University of North Carolina, 1951.

[24] Maynez J., Narayan S., Bohnet B., McDonald R., *On Faithfulness and Factuality in Abstractive Summarization*. arXiv:2005.00661, Google Research, May 2020.

[25] Pragati Baheti, *A Comprehensive Guide to Convolutional Neural Networks*. V7Labs, Microsoft, 2021.

[26] McCloskey M., J. Cohen N., *Catastrophic Interference in Connectionist Networks: The Sequential Learning Problem*. Academic Press, Psychology of Learning and Motivation, vol. 24, pp. 109-165, 1989.

[27] Jin H., Wu D., Zhang S., Zou X., Jin S., Tao D., Liao Q., Xia W., *Design of a Quantization-based DNN Delta Compression Framework for Model Snapshots and Federated Learning*. Washington State University, EECS, 2023.

[28] Agrawal A., Reddy S., Bhattamishra S., Prabhakara Sarath Nookala V., Vashishth V., Rong K., Tumanov A., *DynaQuant: Compressing Deep Learning Training Checkpoints via Dynamic Quantization*. arXiv:2306.11800, Georgia Institute Of Technology, University of Oxford, 2023.

[29] Li W., Chen X., Shu H., Tang Y., Wang Y., *ExCP: Extreme LLM Checkpoint Compression via Weight-Momentum Joint Shrinking*. arXiv:2406.11257, Huawei, 2024.

[30] Krause J., Jin H., Yang J., Fei-Fei L., *Fine-Grained Recognition without Part Annotations*. arXiv:1702.01721, Adobe Research, Stanford, 2013

[31] Loshchilov I., Hutter F., *Decoupled Weight Decay Regularization*. arXiv:1711.05101, ICLR, 2019.