Deep Fusion: Efficient Network Training via Pre-trained Initializations

Hanna Mazzawi [1] **Xavi Gonzalvo** [1] **Michael Wunder** [1]

Abstract

In recent years, deep learning has made remarkable progress in a wide range of domains, with a particularly notable impact on natural language processing tasks. One of the challenges associated with training deep neural networks is the need for large amounts of computational resources and time. In this paper, we present Deep Fusion, an efficient approach to network training that leverages pre-trained initializations of smaller networks. We show that Deep Fusion accelerates the training process, reduces computational requirements, and leads to improved generalization performance on a variety of NLP tasks and T5 model sizes. Our experiments demonstrate that Deep Fusion is a practical and effective approach to reduce the training time and resource consumption while maintaining, or even surpassing, the performance of traditional training methods.

1. Introduction

Large language models (LLMs) have significantly advanced the state of the art in various natural language processing (NLP) tasks, including text generation, translation, summarization, and question answering. However, training these models demands substantial amounts of data and computational resources. As a result, there has been a growing interest in developing efficient training methods to address the challenges associated with the high computational costs and energy consumption during the training process (17).

While some studies (13; 22; 28) discuss that a balance of data and model size is important, it's undeniable that larger models often yield better performance (3). Several experiments and publications have demonstrated that as model size increases, the performance on various natural language processing tasks continues to improve (4; 18; 2). This trend is evident in the progression of LLMs, such as BERT, GPT-2, GPT-3, and PaLM where each successive generation is larger and achieves better results across a wide range of benchmarks (10).

Advancements in large language model (LLM) efficiency have been driven by a variety of innovative techniques that enable faster training or inference without sacrificing performance. One such approach is model compression, which has been shown to reduce LLM size without significant loss in accuracy (7; 27; 14). Similarly, adaptive computation time methods have been proposed to dynamically allocate computational resources during LLM training, leading to improved efficiency (9). Techniques such as layer-wise adaptive rate scaling (LARS) and layer-wise adaptive rate control (LARC) have demonstrated accelerated convergence in LLMs by adapting learning rates on a per-layer basis (25; 24). Moreover, recent studies have explored the potential of mixed-precision training, where lower-precision computation is employed during the training process to speed up training and reduce memory requirements (15). On top of that, efficient training distribution is a combination of data and model parallelization. Data parallelization splits the training batch across accelerators (e.g., GPUs), while model parallelization splits the model operations across accelerators so that each accelerator compute part of the model.

While data parallelism alone is typically the easiest to implement, it is not well suited for very large models as it needs the whole model to fit in a single accelerator. Model parallelism can be efficient, but it can be more difficult to implement as the dependency between accelerators input and outputs can lead to degraded performance.

In our research, we emphasize training efficiency as a primary goal. Unlike the traditional approach concentrating on discovering pruned networks (11; 5), our approach aims to minimize training time by initializing large networks from training smaller ones. We employ fusion operators to combine these smaller networks, promoting wide over-parameterization.

1.1. Contribution

As part of the deep fusion method, this paper proposes:

- A method that focuses on initializing large networks from training smaller networks, and employing fusion operators to combine them. This method promotes

[1]Google Research, New York. Correspondence to: Hanna Mazzawi <mazzawi@google.com>.

Work presented at the ES-FoMo Workshop at ICML 2023. Copyright 2023 by the author(s).

wide over-parameterization, which leads to improved efficiency in network training.

- An effective framework for the utilization of data and model parallelization techniques, as well as the strategic use of accelerator devices to train models of smaller size. This allows our approach to significantly reduce training time while increasing the performance of the resulting networks.
- A downstream task evaluation with LLMs, demonstrating its effectiveness and efficiency in various scenarios.

2. Related Work

In line with the lottery ticket hypothesis (5; 6), our work shares the following belief: The most commonly used initialization schemes, primarily discovered heuristically (8; 12), are sub-optimal. While there's some evidence that over-parameterization may not be necessary during training (16; 1), we still believe over-parameterization and a "good" initialization can yield better performance. Thus, we aim to actualize some of the potential that comes from finding a more principled initialization scheme.

From a transfer learning perspective, progressive networks (20) grow networks to address the problem of forgetting previous tasks. Another approach, deep model consolidation (26), uses a smaller pre-trained model to provide a better initialization for a larger model, which is then fine-tuned on a new task. Network morphism (23) is another approach that aims to find a larger network by transforming a smaller, pretrained network while preserving the network function during the transformation. This is achieved by expanding the original network with layer-wise operations that preserve input-output behavior.

Similar to our method, staged training (21) also focuses on network efficiency. This approach involves defining a growth operator while preserving constraints associated with loss and training dynamics. By gradually expanding the model capacity staged training allows for more efficient training. We argue that preserving training dynamics might not be the most effective approach when it comes to fusion. In fact, it could be counterproductive, and exploring high learning rate cycles could offer a preferable alternative. Furthermore, we enhance the fuse operator by developing a more efficient initialization for cross-connections.

3. Fusion

We start by demonstrating our FUSION operator on two fully connected layers before expanding to T5 transformers.

A generic neural network is a function $f\colon \mathbb{R}^d \to \mathbb{R}^k$ defined with L layers with weights in layer $k \in [L]$ being w_k and biases being b_k. That is, For each layer k we calculate

$$a_k = h_k(a_{k-1}) = g_k(a_{k-1}w_k + b_k), \tag{1}$$

where $a_0 = x$ is the input vector, and g_k is the kth activation function. In what follows, we will omit a_k when it is clear from context.

The output of the neural network is defined as the composition of the L layers,

$$f(x) = h_L \circ \ldots \circ h_2 \circ h_1(x). \tag{2}$$

Our FUSION operator F takes two layers from two different models and generates a new layer by composing their weights and biases. The fused layer has two characteristics:

- **Fusion rule**: the fused layer maintains the same composition or architecture defined in Eq.1. That is, we do not allow a change in the architecture, but rather a change in the dimensionality of the operations.
- **Fusion property**: the fused layer calculates the concatenation of the the two original layers that are fused.

The FUSION operator is defined as follows. Given two layers with d, d' inputs and k, k' outputs,

$$F_w\colon \mathbb{R}^{d\times k} \times \mathbb{R}^{d'\times k'} \to \mathbb{R}^{(d+d')\times(k+k')}, \tag{3}$$
$$F_b\colon \mathbb{R}^{k} \times \mathbb{R}^{k'} \to \mathbb{R}^{(k+k')}. \tag{4}$$

The FUSION of the weights performed by F_w results in a new matrix where the weights of the layers of the two models are located in the diagonal and the rest is set to zero. Similarly, the new bias is simply the concatenation of the bias of the two layers being fused. So the new fused weight $w^{(f)}$ and new bias $b^{(f)}$ taking the weights of two layers, w, w', and bias b, b', respectively is defined as,

$$w^{(f)} = \begin{pmatrix} w & \vec{0} \\ \vec{0} & w' \end{pmatrix}, \qquad b^{(f)} = [b, b'], \tag{5}$$

where $\vec{0}$ is the zero matrix. The output of the fused layer k is defined as,

$$\begin{aligned} h_k^{(f)} =& F(h_k, h'_k) = g_k(a_{k-1}^{(f)} F_w(w_k, w'_k) + F_b(b_k, b'_k)) \\ =& g_k\left([a_{k-1}, a'_{k-1}] \begin{pmatrix} w_k & \vec{0} \\ \vec{0} & w'_k \end{pmatrix} + [b_k, b'_k]\right) \\ =& g_k([a_{k-1}w_k + b_k, a'_{k-1}w'_k + b'_k]) = [h_k, h'_k]. \end{aligned}$$

This means that the result of the FUSION operator on two layers is the concatenation of the outputs, that is $[h_k, h'_k]$.

3.1. Deep Fusion and Self Deep Fusion

For two neural networks f and f' defined as in Eq. 2, the deep fusion of the two models is defined as follows. Denote by, $L(f, f') = F(h_L, h'_L) \circ \ldots \circ F(h_1, h'_1)([x, x])$; And by $\textsc{Avg}(x, y) = (x + y)/2$. The function that averages two vectors of the same dimension, then the deep fusion is defined as,

$$DF(f, f') = \textsc{Avg}(L(f, f')).$$

Intuitively, the deep fused model is maintaining a concatenation of the hidden representations from models f and f' (fusion property) throughout the network, and taking the average of their logits.

This means that after the deep fusion operation, the function calculated by the model is equivalent to the function of average ensemble of the two models. However, if we continue training the fused model, the extra parameters added by the zero blocks in the example can start leveraging the hidden representation from the cross model, and potentially lead to better performance.

Deep fusion allows the models to be distributed across multiple GPUs while still taking advantage of the strengths of both data parallelism and model parallelism.

Self deep fusion of a model f is defined as deep fusing the model with itself (that is, $DF(f, f)$). It can be thought of as a growth operation that does not change the network's predictions to any given input.

3.2. Deep Fusing T5 Transformers

This section describes how to deep fuse two (or more) T5 models (19), f and f', discussing the particularities of each layer type. Once the fusion is completed, the hidden representation of the newly fused model should be a combination of the two hidden representations from the original models, aligned along the feature dimension axis.

Starting from the bottom layer, the fusion of the embedding layer is trivial. Next, for the multi-head attention, if f has y heads, and f' has y' heads, then, the fused model will have $y + y'$ heads. All projections (query, key, value, attention output) as well as the MLP blocks are treated to prevent leaking information from the wrong hidden representation at initialization.

Note that skip connections and activations are parameter free and do not need further handling. Similarly, the element-wise scaling operation holds a scaling parameter per element in the hidden representation, and thus is trivial to fuse.

Lastly, the fusion of the normalization of the hidden representation between attention and MLP layers proves to be unfeasible. This is due to the fact that it's not possible to uphold the fusion rule and the fusion property simultaneously. For the normalization layer we either: 1) Preserve the fusion property but break the fusion rule by normalizing the hidden representations of the sub-models individually and then concatenating them; or 2) keep the fusion rule but violate the fusion property by treating the concatenated hidden representation as a single vector for normalization. It's important to note that the first option requires additional coding beyond parameter initialization, unlike the second option. This dilemma doesn't occur in self deep fusion.

4. Experiments

We begin by training T5 language models on the C4 dataset. The term 'deep' will be dropped when context allows.

4.1. Language Models

The aim of this experiment is to establish a better initial checkpoint for a large T5 (19) transformer network, referred to as T5-MEDIUM, by using two smaller T5 models, denoted as T5-SMALL. We present two types of results: fusing two unique small models and fusing one model with itself (self fusion). We trained the following 4 experiments (see dimensionalities in Table 6 in Appendix A):

1. `baseline`: T5-MEDIUM from random initialization.
2. `fusion-rule`: T5-MEDIUM trained from fusing the two T5-SMALL models while maintaining the fusion rule.
3. `fusion-prop`: T5-MEDIUM trained from fusing the two T5-SMALL models while maintaining the fusion property.
4. `self-fusion`: T5-MEDIUM trained from self fusing a T5-SMALL model.

Zero matrices in Eq. 5 were substituted with blocks initialized randomly with a low variance. Final results are displayed in Table 1 and Figure 1 shows the evaluation metric curves throughout the training.

Model	Loss @1M	Accuracy @1M
`baseline`	4.66e+4	66.65 ± 0.01
`fusion-rule`	4.61e+4	66.88
`fusion-prop`	**4.53e+4**	**67.25** ± 0.03
`self-fusion`	4.55e+4	67.20 ± 0.05

Table 1. Performance of different T5-Medium fusion methods at 1 million steps, replicated three times for standard deviation.

The outcomes of our experiments indicate that while it requires extra code changes to the T5 transformer, upholding the fusion property results in superior performance compared to adhering to the fusion rule. Furthermore, we discovered that self fusion yields comparable performance to standard fusion. Significantly, the `baseline` required an additional 860K steps to achieve the performance level of self fusion. When employing self fusion, training a T5-MEDIUM *resulted in an 18% reduction in computation time* compared to the `baseline`. [1]

[1]T5-SMALL model training time included.

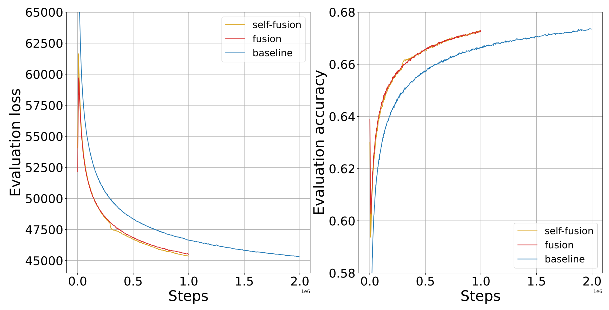

Figure 1. Accuracy and loss on validation data.

4.2. Fusion in Stages

We explored staged fusion using T5-S, T5-M, and T5-L architectures (Table 7, Appendix B) and tested various fusion settings depicted in Figure 2.

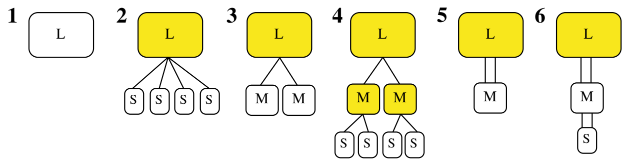

Figure 2. Settings for final T5-L fusion: yellow signifies fused models, white indicates regular training, and links represent fusion (double link signifies self fusion).

Every model (T5-S, T5-M, T5-L) is trained 1M steps. Table 2 below present the performance of the various models.

Model	Loss @1M steps	Accuracy @1M steps
(1)	4.04e+4	69.89
(2)	3.93e+4	70.45
(3)	3.9e+4	70.56
(4)	**3.87e+4**	**70.74**
(5)	3.91e+4	70.57
(6)	3.91e+4	70.47

Table 2. Performance of the various ways of fusing T5-L.

The results show similar performance between fusion and self fusion (settings (3) and (5)). However, repeated self fusion reduces performance, while multiple regular fusions enhance T5-L performance.

Training a model using a single application of self fusion, setting (5), results in a *20% reduction in computation time* compared to the standard setting (1).

4.3. Fine Tuning for Down Stream Tasks

We fine-tuned high performing settings from the first experiment together with a baseline on NLP tasks using the GLUE benchmark. We trained two T5-SMALL models for 500K steps before fusing and self fusing them to create a T5-MEDIUM. We also trained a standalone T5-MEDIUM. These models were fine-tuned at 0 (`baseline` vs fusion without extra training), 250K, 500K, and 1M steps (`baseline` only). The GLUE average results are shown in Table 3 and Figure 3. The complete results for each task is presented in Appendix C.

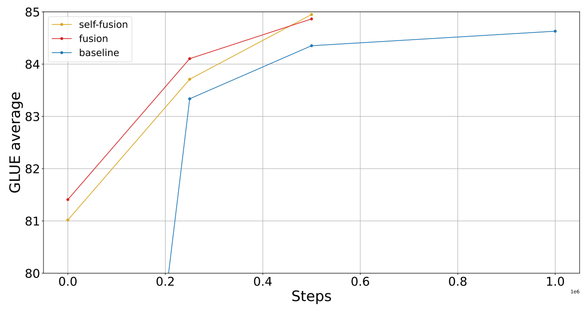

Figure 3. Performance (Glue average - an average over many NLP tasks that score between 0 and 100) of the various models.

Model / step	0	250K	500K	1M
`baseline`	64.07	83.33	84.35	84.74±0.13
`fusion-prop`	**81.40**	**84.10**	84.86±0.13	-
`self-fusion`	81.01	83.71	**84.94**±0.2	-
T5-SMALL	-	-	-	80.28

Table 3. Performance (GLUE average) of the various models on downstream tasks, replicated three times for standard deviation.

Our results indicate that enhancing a pretrained model's performance may simply require self-fusion before fine-tuning, without further pretraining. For instance, a T5-SMALL model, trained for 500K steps, when self-fused and fine-tuned, outperforms the same model trained to 1M steps before fine-tuning (81.01 vs 80.28). It's evident that the extra parameters from self-fusion benefit NLP tasks more than extended pretraining.

Next, the results above also suggest that deep fusion can lead to faster training to better performance, when fine-tuning on downstream NLP tasks. However, while in pretrain, the training curves of fusion of self fusion look similar, we can see that for downstream tasks, fusion maintain higher performance throughout till convergence (in here, both models converge to similar performance).

Model / time	Fusion	Post fusion	Time	GLUE
`baseline`	0 steps 0h	1M steps 39.2h	39.2h	84.74
`fusion-prop`	500k steps 2×8h	500k steps 21.9h	37.9h	84.86
`self-fusion`	500k steps 8h	500k steps 21.9h	**29.9h**	**84.94**

Table 4. Compute time in hours (TPU V3 4x4 topology).

The total compute saving is about 24% TPU time for this configuration as presented in Table 4. Even though we trained for less time, the final performance was slightly better than the baseline.

5. Discussion and Conclusion

In this paper, we present a new technique for improving the training process of large models. Our technique, called deep fusion, combines multiple models into a single model that can be trained more efficiently. We demonstrate how model fusion can be used to reduce the restrictions of distributed training, save on overall compute costs, and improve model performance.

In our experiments we fused models that are trained on the same data and have identical architectures. While fusion has immediate training advantages, further research is needed to understand the implications and possible applications of fusing models trained on different sources and distinct architectures.

For example, it would be interesting to explore if transfer learning occurs when fusing models trained in different domains. Additionally, it would be interesting to understand the characteristics of models that are the fusion of models that differ in dimensionality. For example, one model could be attention-heavy, while another could be MLP-heavy. Finally, it would be interesting to explore model fusion when the models are trained on different sequence lengths. This could also lead to efficiency improvements, as lower-length models train faster.

We believe that model fusion is a promising technique for improving the training process of large models. We hope that our work will inspire further research in this area.

References

[1] Mikhail Belkin, Daniel Hsu, Siyuan Ma, and Soumik Mandal. Reconciling modern machine-learning practice and the classical bias-variance trade-off. *Proceedings of the National Academy of Sciences*, 116(32):15849–15854, 2019.

[2] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, pages 14182–14193, 2020.

[3] Aakanksha Chowdhery, Sharan Narang, Jacob Devlin, Maarten Bosma, Gaurav Mishra, Adam Roberts, Paul Barham, Hyung Won Chung, Charles Sutton, Sebastian Gehrmann, Parker Schuh, Kensen Shi, Sasha Tsvyashchenko, Joshua Maynez, Abhishek Rao, Parker Barnes, Yi Tay, Noam Shazeer, Vinodkumar Prabhakaran, Emily Reif, Nan Du, Ben Hutchinson, Reiner Pope, James Bradbury, Jacob Austin, Michael Isard, Guy Gur-Ari, Pengcheng Yin, Toju Duke, Anselm Levskaya, Sanjay Ghemawat, Sunipa Dev, Henryk Michalewski, Xavier Garcia, Vedant Misra, Kevin Robinson, Liam Fedus, Denny Zhou, Daphne Ippolito, David Luan, Hyeontaek Lim, Barret Zoph, Alexander Spiridonov, Ryan Sepassi, David Dohan, Shivani Agrawal, Mark Omernick, Andrew M. Dai, Thanumalayan Sankaranarayana Pillai, Marie Pellat, Aitor Lewkowycz, Erica Moreira, Rewon Child, Oleksandr Polozov, Katherine Lee, Zongwei Zhou, Xuezhi Wang, Brennan Saeta, Mark Diaz, Orhan Firat, Michele Catasta, Jason Wei, Kathy Meier-Hellstern, Douglas Eck, Jeff Dean, Slav Petrov, and Noah Fiedel. Palm: Scaling language modeling with pathways, 2022.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 4171–4186, 2018.

[5] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *Proceedings of the International Conference on Learning Representations*, 2018.

[6] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis at scale. In *arXiv preprint arXiv:1903.01611*, 2019.

[7] Prakhar Ganesh, Yao Chen, Xin Lou, Mohammad Ali Khan, Yin Yang, Hassan Sajjad, Preslav Nakov, Deming Chen, and Marianne Winslett. Compressing Large-Scale Transformer-Based Models: A Case Study on BERT. *Transactions of the Association for Computational Linguistics*, 9:1061–1080, 09 2021.

[8] Xavier Glorot and Yoshua Bengio. Understanding the difficulty of training deep feedforward neural networks. In *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, pages 249–256, 2010.

[9] Alex Graves. Adaptive computation time for recurrent neural networks. In *Proceedings of the International Conference on Learning Representations*, 2016.

[10] Jiatao Gu, Jianqiang Hu, Tong Zhao, Ying Lin, Xiuying Cheng, Lijun Wang, and Xiang Wan. Palm: Pre-training an autoencoding and autoregressive language model for context-conditioned generation. In *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pages 2643–2662, 2021.

[11] Song Han, Jeff Pool, John Tran, and William J. Dally. Learning both weights and connections for efficient neural networks. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 1*, NIPS'15, page 1135–1143, Cambridge, MA, USA, 2015. MIT Press.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE International Conference on Computer Vision*, pages 1026–1034, 2015.

[13] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*, 2020.

[14] Woosuk Kwon, Sehoon Kim, Michael W. Mahoney, Joseph Hassoun, Kurt Keutzer, and Amir Gholami. A fast post-training pruning framework for transformers. In Alice H. Oh, Alekh Agarwal, Danielle Belgrave, and Kyunghyun Cho, editors, *Advances in Neural Information Processing Systems*, 2022.

[15] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, et al. Mixed precision training. In *Proceedings of the International Conference on Learning Representations*, 2017.

[16] Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. In *Proceedings of the International Conference on Learning Representations*, 2020.

[17] Deepak Narayanan, Mohammad Shoeybi, Jared Casper, Patrick LeGresley, Mostofa Patwary, Vijay Korthikanti, Dmitri Vainbrand, Prethvi Kashinkunti, Julie Bernauer, Bryan Catanzaro, Amar Phanishayee, and Matei Zaharia. Efficient large-scale language model training on gpu clusters using megatron-lm. In *Proceedings of the International Conference for High Performance Computing, Networking, Storage and Analysis*, SC '21, New York, NY, USA, 2021. Association for Computing Machinery.

[18] Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Language models are unsupervised multitask learners. In *OpenAI Blog*, 2019.

[19] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, 2019.

[20] Andrei A Rusu, Neil C Rabinowitz, Guillaume Desjardins, Hubert Soyer, James Kirkpatrick, Koray Kavukcuoglu, Razvan Pascanu, and Raia Hadsell. Progressive neural networks. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, pages 2212–2220, 2016.

[21] Sheng Shen, Pete Walsh, Kurt Keutzer, Jesse Dodge, Matthew Peters, and Iz Beltagy. Staged training for transformer language models. In Kamalika Chaudhuri, Stefanie Jegelka, Le Song, Csaba Szepesvari, Gang Niu, and Sivan Sabato, editors, *Proceedings of the 39th International Conference on Machine Learning*, volume 162 of *Proceedings of Machine Learning Research*, pages 19893–19908. PMLR, 17–23 Jul 2022.

[22] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, Aurelien Rodriguez, Armand Joulin, Edouard Grave, and Guillaume Lample. Llama: Open and efficient foundation language models, 2023.

[23] Chen Wei, Haoyu Wang, Yongyang Rui, and Changqing Chen. Network morphism. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning*, pages 2662–2671, 2016.

[24] Yang You, Yaroslav Bulatov, Igor Gitman, and Andrej Risteski. Large batch training of convolutional networks. In *arXiv preprint arXiv:1708.03888*, 2017.

[25] Yang You, Igor Gitman, and Boris Ginsburg. Scaling sgd batch size to 32k for imagenet training. In *Deep Learning Scaling is Predictable, Empirically*, page 13, 2017.

[26] Junting Zhang, Jie Zhang, Shalini Ghosh, Dawei Li, Serafettin Tasci, Larry Heck, Heming Zhang, and C. C. Jay Kuo. Class-incremental learning via deep model consolidation, 2020.

[27] Qingru Zhang, Simiao Zuo, Chen Liang, Alexander Bukharin, Pengcheng He, Weizhu Chen, and Tuo Zhao. Platon: Pruning large transformer models with upper confidence bound of weight importance, 2022.

[28] Chunting Zhou, Pengfei Liu, Puxin Xu, Srini Iyer, Jiao Sun, Yuning Mao, Xuezhe Ma, Avia Efrat, Ping Yu, Lili Yu, Susan Zhang, Gargi Ghosh, Mike Lewis, Luke Zettlemoyer, and Omer Levy. Lima: Less is more for alignment, 2023.

Model	Glue avg	COLA Matthew's	SST acc	MRPC f1	MRPC acc	STS-b pearson	STS-b spearman	qqp acc	qqp f1	MNLI-m	MNLI-mm	QNLI	RTE
baseline 1m	84.74	54.18	94.38	93.17	90.69	89.83	89.75	91.94	89.13	86.77	86.6	92.13	78.34
baseline 1m	84.45	52.95	93.92	93.12	90.44	89.49	89.35	91.94	89.14	86.94	86.58	92.26	77.98
baseline 1m	84.69	53.15	93.81	92.15	88.97	89.06	88.93	92.04	89.24	86.55	86.21	91.69	82.31
Fusion 500k	84.98	53.97	94.27	92.91	90.2	89.68	89.49	92.04	89.36	86.6	86.43	92.39	80.87
Fusion 500k	84.68	54.46	93.81	91.8	88.97	89.44	89.28	91.95	89.13	86.67	86.65	92.11	80.14
Fusion 500k	84.92	53.94	94.5	92.73	89.71	90.62	90.44	92.01	89.23	86.64	86.45	91.56	80.51
Self fusion 500K	84.69	54.58	93.12	92.03	89.22	89.23	89.16	91.81	89.01	86.64	86.68	92.11	80.87
Self fusion 500K	85.19	55.82	93.69	93.1	90.44	89.9	89.73	92.04	89.34	86.8	86.31	92.31	80.87
Self fusion 500K	84.95	58	94.27	92.36	89.71	89.93	89.7	91.96	89.18	86.47	86.4	91.93	77.62

Table 5. Performance (Glue tasks) of the various models on downstream tasks.

Appendix A

In this appendix, we list the dimension of the T5 transformers used in the first experiment.

Model Name	T5-Small	T5-Medium
embedding dim	512	1024
number of heads	6	12
enc./dec. layers	8	8
head dim	64	64
mlp dimension	1024	2048
number of parameters	77M	242M

Table 6. Dimensions of T5 Small and Medium.

Appendix B

In this appendix, we list the dimension of the T5 transformers used in the second experiment.

Model Name	T5-S	T5-M	T5-L
embedding dim	512	1024	2048
number of heads	6	12	24
enc./dec. layers	8	8	8
head dim	128	128	128
mlp dimension	1024	2048	4096
number of parameters	95M	317M	1.1B

Table 7. Dimensions of T5-S, T5-M and T5-L.

Appendix C

In this Appendix we list the full results (see Table 5) of the downstream on various Glue tasks. The average is calculated over all tasks. For tasks with more than one metrics, we average the metrics and then average over the tasks.