SuperShaper: A Pre-Training Approach for Discovering Efficient Transformer Shapes

Vinod Ganesan [*1 2] **Gowtham Ramesh** [*3 4] **Pratyush Kumar** [1 2 3] **Raj Dabre** [3 5]

Abstract

Task-agnostic pre-training followed by task-specific fine-tuning is a default approach to train NLU models which need to be deployed on devices with varying resource and accuracy constraints. However, repeating pre-training and fine-tuning across tens of devices is prohibitively expensive. To address this, we propose SuperShaper, a task-agnostic approach wherein we pre-train a single model which subsumes a large number of Transformer models via linear bottleneck matrices around each Transformer layer which are sliced to generate differently shaped sub-networks. Despite its simplicity, SuperShaper radically simplifies NAS for language models and discovers networks, via evolutionary algorithm, that effectively trade-off accuracy and model size. Discovered networks are more accurate than a range of hand-crafted and automatically searched networks on GLUE benchmarks. Further, a critical advantage of shape as a design variable for NAS is that the networks found with these heuristics derived for good shapes, match and even improve on carefully searched networks across a range of parameter counts.

1. Introduction

While the pre-train then fine-tune paradigm (Devlin et al., 2019) has been very successful in developing state-of-the-art NLP models, it is very compute intensive. This limits the personalization of models for the product space of multiple tasks and devices. Super pre-training a model subsuming a family of models instead of individually pre-training them, as performed in NAS-BERT (Xu et al., 2021), is the most attractive solution. However, existing NAS-like approaches utilise a complex design space with many exploration dimensions. In this work, we set out to understand if there is a simple characterization of the family of efficient networks. We propose SuperShaper, a simple and elegant method for super pre-training language models. It differs from existing super pre-training efforts (Hou et al., 2020; Zhang et al., 2021; Xu et al., 2021) in its characterisation of design space. Our simple characterization is the notion of a "shape" of a network, which we define as the hidden dimensions of each transformer layer enabled via bottleneck matrices at the input and output of each layer (inspired by MobileBERT (Sun et al., 2020)). These differently shaped networks are randomly sampled by slicing the bottleneck matrices and trained, similar to earliest proposed elastic NAS efforts (Brock et al., 2018; Cai et al., 2019). SuperShaper can be enabled for any transformer model with just 20 lines of PyTorch code.

In order to choose optimal shapes we leverage Evolutionary Algorithm (EA) to search over SuperShaper's shape space. This yields shapes that are competitive on GLUE tasks with BERT-base as well as other compressed models (both hand-crafted and NAS). Analysis of discovered networks across parametric counts helps identify heuristics of good shapes, which we call the pipe rule. By applying our pipe rule, we hand-craft sub-networks which match or often exceed the performance of networks discovered using EAs. These hand-crafted sub-networks suggest a *cigar-like* shape. Thus, our technique indicates that NAS can be performed with radically simpler design spaces which generalise across tasks and parameter counts.

2. Related Work

Three dominant strategies define efficient deployment of language models: (a) model compression with low-rank approximation, pruning, and quantization (Wang et al., 2019; Ma et al., 2019; Michel et al., 2019; Goyal et al., 2020; Shen et al., 2020), (b) task-agnostic and task-specific knowledge distillation (Sanh et al., 2019a; Jiao et al., 2020; Sun et al., 2020; Wang et al., 2020b; Jiao et al., 2020; Tang et al., 2019;

[*]Equal contribution [1]Microsoft Research, India [2]IIT Madras, India [3]Nilekani Centre at AI4Bharat [4]University of Wisconsin, Madison [5]National Institute of Information and Communications Technology, Kyoto, Japan. Correspondence to: Vinod Ganesan <vinodg@cse.iitm.ac.in>.

Proceedings of the 40^{th} International Conference on Machine Learning, Honolulu, Hawaii, USA. PMLR 202, 2023.

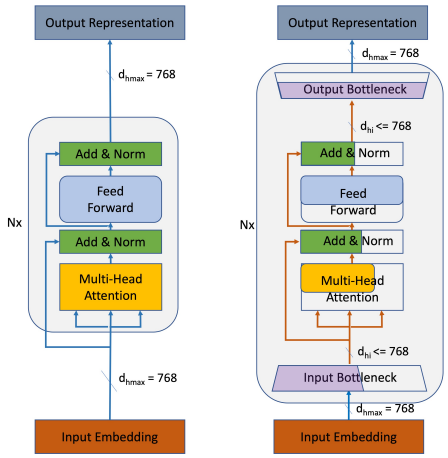

Figure 1. A Transformer layer in (a) BERT, and (b) Backbone in SuperShaper with bottleneck matrices.

Turc et al., 2019a; Sun et al., 2019a; Chen et al., 2020a), and (c) Neural Architecture Search (Gao et al., 2021; Chen et al., 2020b; Xu et al., 2021; Zhang et al., 2021).

In contrast, SuperShaper provides a radically simple NAS method to super pre-train a family of models enabling the discovery of efficient models across the product space. Model compression and knowledge-distillation are complementary to our effort and can be composed together for further gains. In addition, SuperShaper uses a very simple design space unlike other contemporary efforts (Xu et al., 2021; Zhang et al., 2021).

3. SuperShaper Methodology

SuperShaper methodology involves shape variable pre-training, subnetwork search and fine-tuning.

3.1. Architecture and Pre-Training

We focus on the Transformer architecture (Vaswani et al., 2017) used to train BERT-like encoders (see Figure 1). With SuperShaper, we would like to explore subnetworks where layers have different hidden dimensions. The intuition behind this choice is that different layers may perform roles of varying importance. For instance, earlier layers manipulate the input embeddings and the final layers responsible for the output may require larger hidden dimensions. Inspired by MobileBERT (Sun et al., 2020), we modify the standard Transformer layer by inserting bottleneck layers before and after each layer, as shown in Figure 1 (b). We slice the bottleneck matrices at the input to project vectors to a smaller dimension, and then at the output to project them back to the original dimension. The weight matrices of the transformer sublayers are appropriately sliced to process the down-projected vectors. With this simple change, we can generate differently shaped subnetworks for super-pretraining.

We denote the SuperShaper backbone as T and any sub-network sliced from the backbone as T_S where S is the *shape* vector that represents the layer-wise hidden dimensions, S_i for layer i. The set of all possible values of S denotes the design space D. The smallest and largest sub-networks in D are denoted as T_{S^-} and T_{S^+}, respectively, while a random sub-network is denoted as T_{S^r}.

From a given design space D, we randomly *sample* n different shapes S and obtain T_S for each by the slicing technique described in the previous subsection. Additionally consider the largest and smallest sub-networks T_{S^+} and T_{S^-}, also called the *sandwich rule* (Yu and Huang, 2019) which has been shown to perform better for weight-sharing NAS in computer vision (Yu and Huang, 2019; Yu et al., 2020; Wang et al., 2021). The sampled sub-networks, are jointly trained with the masked language model (MLM) objective.

3.2. Searching for optimal shapes

Once we have super pre-trained with a design space D, we can sample T_S for any S, which can then be fine-tuned. The design space of all sub-networks can be large: A choice of 7 shapes each for 12 layers can yield 14 billion sub-networks. To find an optimal shape from S which meets specific constraints on accuracy, parameter count, or latency on devices, we adopt Evolutionary Algorithm (EA) from (Real et al., 2017) which starts with a population of solutions and over generations create new solutions by applying genetic operations like mutation and crossover and retain the fittest solutions based on defined metrics of interest. For SuperShaper, the genetic representation of sub-networks and genetic operations are natural and simply described by the shape vector S. For the fitness metrics, we use perplexity on language modelling and latency on a device. To amortize the expense of computing these metrics for thousands of solutions, we use fitness predictors that have been studied elsewhere in NAS (Cai et al., 2019; Ganesan et al., 2020).

3.3. Fine-tuning T_S from SuperShaper

A sampled sub-network T_S can be fine-tuned via: direct fine-tuning (), further pre-training and then fine-tuning (), and pre-training T_S from scratch before fine-tuning (). The main results are reported for with further analysis on others in Appendices C and G.

4. Experimental Setup and Results

We now detail the experimental setup and report a range of findings to evaluate SuperShaper.

4.1. Experimental Setup

We describe the experimental setups for pre-training and fine-tuning. For details see Appendix A and B.

SuperShaper pre-training We insert randomly initialized bottleneck matrices into BERT-base and slice them to produce Transformer layers of varying hidden dimensions in $\{120, 240, 360, 480, 540, 600, 768\}$. Thereafter we follow the training procedure in Section 3.1 using the MLM objective on the C4 RealNews dataset (Raffel et al., 2019) with a batch size of 2048, with max seq. length of 128, and 175K steps (or 26 epochs) on 8 A100 GPUs.

Evolutionary Algorithm (EA). We adapt the EA presented in (Real et al., 2017) with population size 100, and mutation prob 0.4 bounded by 300 iterations.

Fitness Predictors. We randomly sample $10,000$ sub-networks and build XGBoost-based (Chen et al., 2015) fitness predictor models to predict perplexity on the C4-RealNews validation dataset. These are used in the aforementioned EA.

Fine tuning. We use standard hyperparameters for fine-tuning on GLUE (Wang et al., 2018) and Squad V1 (Rajpurkar et al., 2016). For the GLUE tasks RTE, MRPC and STS-B, we start with a model fine-tuned on MNLI similar to (Liu et al., 2019). We only hyperparameter tune for batch size, weight decay, and warmup steps and report results on best ones.

4.2. Main Results

Comparing with BERT-base. Using EA, we discover a 96M parameter sub-network from SuperShaper and compare it with BERT-base (Xu et al., 2021) with 110M parameters. We witness an equal GLUE score of 83.7% for lesser parameters and a comparable Squad v1 score of 88.2 showcasing the richness of SuperShaper.

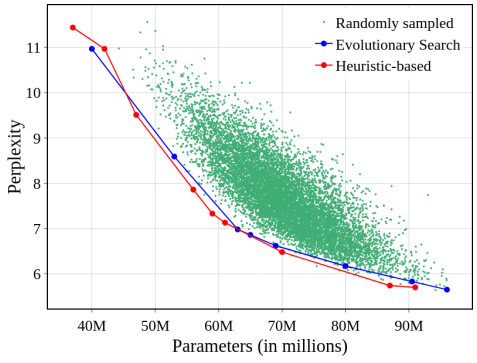

Figure 2. Evolutionary search finds optimal models while simple heuristics yield competitive models.

Comparing with compressed models. We now compare SuperShaper models with state-of-the-art compressed models (both hand-crafted and NAS-discovered) within 60-67M parameters range. The task-wise performance of the obtained sub-network is reported in Table 1. On GLUE, our 63M sub-network performs competitively across the board despite a simpler design space and even outperforms many prominent hand-crafted networks. We derive similar insights in SQuAD. NAS-BERT reports a higher GLUE and EM/F1 scores, potentially attributed to the complex design space with novel separable convolutions. Note, NAS-BERT and DynaBERT use knowledge distillation and data augmentation that are complementary to our approach and can be combined for better performance. In summary, we establish that SuperShaper with a simple design space and efficient super pre-training implementation performs competitively in compressing models to a given parameter count.

Comparing sub-networks with varying sizes. We use EA to search for sub-networks ranging from 40 to 110M parameters. To understand the effectiveness of EA search, we sample 10,000 random sub-networks and compute their perplexity. We then plot these points along with the networks searched by EA in Figure 2. We make two key observations: (a) sub-network's shape critically affects perplexity, and (b) EA effectively discovers accurate networks across the spectrum. We report GLUE scores for these networks in Appendix G.3.

4.3. Shape analysis of Super-Networks.

Given SuperShapers's simple design space, we can ask the question: Are there *good shapes* for different model sizes?

Performance of templated shapes. We first evaluate the performance of the following templated shapes in the 63-65M parameter range: lower triangle (hidden sizes increase from lowest to highest layers), upper triangle (inverse of lower triangle), rectangle (same hidden sizes in all layers), diamond (hidden sizes increase up to middle layer and then shrink), inverted diamond (inverse of diamond), bottle (same as diamond but same hidden sizes in middle layers), and inverted bottle (inverse of bottle). Details of the hidden dimensions and sub-network perplexity for each network are in Appendix G.1. We observe that lower triangle has the lowest perplexity (7.31) while inverted bottle (9.22) has the highest indicating shape's strong impact.

Feature importance from optimal sub-networks. From the analysis above and of sub-networks searched by EA, we find that accurate networks have more parameters in the later layers. We analyse this using a predictor trained to estimate perplexity given the shape. For this predictor, we compute the feature importance (plot in Appendix E) of each layer's shape and derive a set of heuristics of good shapes, which we call pipe rule. Concretely, the pipe rule is as follows: allocate the largest dimension to the last layer,

Model	Params	MNLI-m	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	Avg. GLUE	SQuAD V1
LayerDrop (Fan et al., 2019)	66M	80.7	88.3	88.4	45.4	90.7	-	65.2	85.9	-	-
DistilBERT (Sanh et al., 2019b)	66M	82.2	88.5	89.2	51.3	91.3	86.9	59.9	87.5	79.6	79.1 / 86.9
Bert-PKD (Sun et al., 2019b)	66M	81.5	70.7	89.0	-	92.0	-	65.5	85.0	-	77.1 / 85.3
MiniLM (Wang et al., 2020b)	66M	84.0	91.0	91.0	49.2	92.0	-	71.5	88.4	-	-
Ta-TinyBert (Jiao et al., 2020)	67M	83.5	90.6	90.5	42.8	91.6	86.5	72.2	88.4	80.8	-
Tiny-BERT (Jiao et al., 2020)	66M	84.6	89.1	90.4	51.1	93.1	83.7	70.0	82.6	80.6	79.7 / 87.5
BERT-of-Theseus (Xu et al., 2020)	66M	82.3	89.6	89.5	51.1	91.5	88.7	68.2	-	-	-
PD-BERT (Turc et al., 2019b)	66M	82.5	90.7	89.4	-	91.1	-	66.7	84.9	-	-
ELM (Jiao et al., 2021)	67M	84.2	91.1	90.8	54.2	92.7	88.9	72.2	89.0	82.9	77.2 / 85.7
NAS-BERT∗ (Xu et al., 2021)	60M	83.3	90.9	91.3	55.6	92.0	88.6	78.5	87.5	83.5	80.5 / 88.0
DynaBERT† (Hou et al., 2020)	60M	84.2	91.2	91.5	56.8	92.7	89.2	72.2	84.1	82.8	-
YOCO-bert (Zhang et al., 2021)	59-67M	82.6	90.5	87.2	59.8	92.8	-	72.9	90.3	-	-
SuperShaper (ours)	63M	82.2	90.2	88.1	53.0	91.9	87.6	79.1	89.5	82.7	78.25 / 86.01
SuperShaper heuristic-shaped (ours)	61M	82.0	90.3	88.4	52.6	91.6	87.8	77.6	86.5	82.1	77.86 / 85.83

Table 1. Comparison of SuperShaper with 60-67M parameter constraint models on development set of GLUE. † trained with data augmentation, ∗ trained without knowledge distillation during fine-tuning

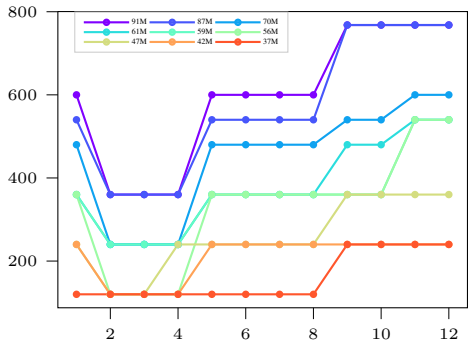

Figure 3. Hidden sizes (y-axis) at different layers (x-axis) for heuristically shaped models resemble cigars.

followed by first, then early middle layers (2-5), and finally later middle layers (6-11). We denote this as a *cigar-like shape*.

Heuristically shaped models. Using pipe rule, we hand-craft subnetworks for a given parameter range (say 60-65M) with cigar-like shapes as shown in Figure 3 and evaluate them on perplexity (for). We find that the performance is competitive and even outperforms sub-networks discovered with EAs (see Figure 2). We also take one of the models built using pipe rule (for 61M) and pre-train and fine-tune them on GLUE (see Table 1). Similar to our subnetwork identified through EA (63M), the pipe rule model outperforms prominent hand-crafted or compressed networks. This strongly demonstrates the generalization of the derived heuristics across model size. To the best of our knowledge, this is the first such generalization demonstrated for NAS.

5. Conclusions

We proposed SuperShaper, a simple and elegant pre-training approach to train a family of language models by varying shapes of Transformer layers to identify optimal shapes, given a budget, via Evolutionary Algorithms. We obtained networks that are competitive with state-of-the-art model compression techniques on GLUE benchmarks. We discovered that cigar-like shapes of networks generalize across parameter counts is insensitive to shape. Consequently, we demonstrated that NAS can be performed with radically simple design space and implementation, while deriving generalization across tasks and model sizes.

6. Limitations

We identify the following limitations of our work: (a) Since subtransformer slices are randomly sampled during super pre-training, there is a possibility that optimal slices may not be fully trained. Additionally, given the billion choices evolutionary search may not always find the best models, (b) Some sub-transformers (smaller prefixes) may get significantly more gradient updates than others and their overall effect on super pre-training needs to be evaluated, and (c) Best downstream performance is obtained from rather than or (it is noteworthy, however, that the latter exhibits competitive performance). This involves additional training effort.

7. Ethical Considerations

We do not create any datasets or collect annotations in this paper. We have used publicly available datasets and models for super pre-training and fine-tuning and thus the models

will contain biases. Debiasing techniques will help resolve these issues but that is beyond the scope of this work.

References

Andrew Brock, Theo Lim, JM Ritchie, and Nick Weston. 2018. Smash: One-shot model architecture search through hypernetworks. In *International Conference on Learning Representations*.

Han Cai, Chuang Gan, Tianzhe Wang, Zhekai Zhang, and Song Han. 2019. Once-for-all: Train one network and specialize it for efficient deployment. In *International Conference on Learning Representations*.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020a. Adabert: Task-adaptive bert compression with differentiable neural architecture search. In *Proceedings of the Twenty-Ninth International Joint Conference on Artificial Intelligence, IJCAI-20*, pages 2463–2469. International Joint Conferences on Artificial Intelligence Organization. Main track.

Daoyuan Chen, Yaliang Li, Minghui Qiu, Zhen Wang, Bofang Li, Bolin Ding, Hongbo Deng, Jun Huang, Wei Lin, and Jingren Zhou. 2020b. Adabert: Task-adaptive bert compression with differentiable neural architecture search. *Cell*, 2(3):4.

Tianqi Chen, Tong He, Michael Benesty, Vadim Khotilovich, Yuan Tang, Hyunsu Cho, et al. 2015. Xgboost: extreme gradient boosting. *R package version 0.4-2*, 1(4):1–4.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Bert: Pre-training of deep bidirectional transformers for language understanding. In *NAACL-HLT (1)*.

Angela Fan, Edouard Grave, and Armand Joulin. 2019. Reducing transformer depth on demand with structured dropout. *arXiv preprint arXiv:1909.11556*.

Vinod Ganesan, Surya Selvam, Sanchari Sen, Pratyush Kumar, and Anand Raghunathan. 2020. A case for generalizable dnn cost models for mobile devices. In *2020 IEEE International Symposium on Workload Characterization (IISWC)*, pages 169–180. IEEE.

Jiahui Gao, Hang Xu, Xiaozhe Ren, Philip LH Yu, Xiaodan Liang, Xin Jiang, Zhenguo Li, et al. 2021. Autobert-zero: Evolving bert backbone from scratch. *arXiv preprint arXiv:2107.07445*.

Saurabh Goyal, Anamitra Roy Choudhury, Saurabh Raje, Venkatesan Chakaravarthy, Yogish Sabharwal, and Ashish Verma. 2020. Power-bert: Accelerating bert inference via progressive word-vector elimination. In *International Conference on Machine Learning*, pages 3690–3699. PMLR.

Lu Hou, Zhiqi Huang, Lifeng Shang, Xin Jiang, Xiao Chen, and Qun Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*, volume 33, pages 9782–9793. Curran Associates, Inc.

Xiaoqi Jiao, Huating Chang, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2021. Improving task-agnostic bert distillation with layer mapping search. *Neurocomputing*, 461:194–203.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2020. Tinybert: Distilling BERT for Natural Language Understanding. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: Findings*.

Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

Xindian Ma, Peng Zhang, Shuai Zhang, Nan Duan, Yuexian Hou, Ming Zhou, and Dawei Song. 2019. A tensorized transformer for language modeling. *Advances in Neural Information Processing Systems*, 32:2232–2242.

Paul Michel, Omer Levy, and Graham Neubig. 2019. Are sixteen heads really better than one? *arXiv preprint arXiv:1905.10650*.

Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J Liu. 2019. Exploring the limits of transfer learning with a unified text-to-text transformer. *arXiv preprint arXiv:1910.10683*.

Pranav Rajpurkar, Jian Zhang, Konstantin Lopyrev, and Percy Liang. 2016. SQuAD: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

Esteban Real, Sherry Moore, Andrew Selle, Saurabh Saxena, Yutaka Leon Suematsu, Jie Tan, Quoc V Le, and Alexey Kurakin. 2017. Large-scale evolution of image classifiers. In *International Conference on Machine Learning*, pages 2902–2911. PMLR.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019a. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *ArXiv*, abs/1910.01108.

Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019b. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Sheng Shen, Zhen Dong, Jiayu Ye, Linjian Ma, Zhewei Yao, Amir Gholami, Michael W Mahoney, and Kurt Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 34, pages 8815–8821.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019a. Patient knowledge distillation for BERT model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332, Hong Kong, China. Association for Computational Linguistics.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019b. Patient knowledge distillation for bert model compression. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pages 4323–4332.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2020. MobileBERT: a compact task-agnostic BERT for resource-limited devices. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 2158–2170, Online. Association for Computational Linguistics.

Raphael Tang, Yao Lu, L. Liu, Lili Mou, Olga Vechtomova, and Jimmy J. Lin. 2019. Distilling task-specific knowledge from bert into simple neural networks. *ArXiv*, abs/1903.12136.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019a. Well-read students learn better: The impact of student initialization on knowledge distillation. *ArXiv*, abs/1908.08962.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019b. Well-read students learn better: The impact of student initialization on knowledge distillation. *arXiv preprint arXiv:1908.08962*, 13.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in neural information processing systems*, pages 5998–6008.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel Bowman. 2018. Glue: A multi-task benchmark and analysis platform for natural language understanding. In *Proceedings of the 2018 EMNLP Workshop BlackboxNLP: Analyzing and Interpreting Neural Networks for NLP*, pages 353–355.

Dilin Wang, Meng Li, Chengyue Gong, and Vikas Chandra. 2021. Attentivenas: Improving neural architecture search via attentive sampling. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 6418–6427.

Hanrui Wang, Zhanghao Wu, Zhijian Liu, Han Cai, Ligeng Zhu, Chuang Gan, and Song Han. 2020a. Hat: Hardware-aware transformers for efficient natural language processing. *arXiv preprint arXiv:2005.14187*.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. 2020b. Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*.

Ziheng Wang, Jeremy Wohlwend, and Tao Lei. 2019. Structured pruning of large language models. *arXiv preprint arXiv:1910.04732*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. BERT-of-Theseus: Compressing BERT by Progressive Module Replacing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 7859–7869.

Jin Xu, Xu Tan, Renqian Luo, Kaitao Song, Jian Li, Tao Qin, and Tie-Yan Liu. 2021. Nas-bert: Task-agnostic and adaptive-size bert compression with neural architecture search. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*.

Jiahui Yu and Thomas S Huang. 2019. Universally slimmable networks and improved training techniques. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 1803–1811.

Jiahui Yu, Pengchong Jin, Hanxiao Liu, Gabriel Bender, Pieter-Jan Kindermans, Mingxing Tan, Thomas Huang, Xiaodan Song, Ruoming Pang, and Quoc Le. 2020. Bignas: Scaling up neural architecture search with big single-stage models. In *European Conference on Computer Vision*, pages 702–717. Springer.

Shaokun Zhang, Xiawu Zheng, Chenyi Yang, Yuchao Li, Yan Wang, Fei Chao, Mengdi Wang, Shen Li, Jun Yang, and Rongrong Ji. 2021. You only compress once: Towards effective and elastic bert compression via exploit-explore stochastic nature gradient. *arXiv preprint arXiv:2106.02435*.

A. Fine tuning tasks and Evaluation metrics

We report performance metrics on the `dev` version of the benchmark. For RTE, MRPC and STS-B, we start with a model fine-tuned on MNLI similar to (Liu et al., 2019; Xu et al., 2021). For metrics, we report Matthews correlation for CoLA (Wang et al., 2018), Spearman correlation for STS-B (Wang et al., 2018) and accuracy for all other tasks. For MNLI-m (Wang et al., 2018), we report accuracy on the matched set. For Squad, we report exact match and F1 score. Following (Devlin et al., 2019; Xu et al., 2021; Zhang et al., 2021; Hou et al., 2020), we also exclude the problematic WNLI dataset. For all the datasets in GLUE, we use the official train and dev splits and download the datasets from HuggingFace datasets[1].

B. Training details and Hyperparameters used in SuperShaper

We initialize our backbone with BERT-base-cased model trained on Wikipedia and BookCorpus with identity bottleneck matrices. We then super pre-train the backbone using MLM over the C4 RealNews dataset with effective batch size of 2048, max sequence length 128, for 175K steps (or 26 epochs) on 8 A100 GPUs. The hyperparameters we used for MLM pretraining and finetuning tasks are detailed in Table 2 and Table 3.

Data	C4/RealNews
Max sequence length	128
Batch size	2048
Peak learning rate	2e-5
Number of steps	175K
Warmup steps	10K
Hidden dropout	0
GeLU dropout	0
Attention dropout	0
Learning rate decay	Linear
Optimizer	AdamW
Adam ϵ	1e-6
Adam (β_1, β_2)	(0.9, 0.999)
Weight decay	0.01
Gradient clipping	0

Table 2. Hyperparameters for MLM super pre-training on C4 RealNews. Super pre-training was done on 8 A100 GPUs

	CoLA	Other GLUE tasks	Squad V1
Batch size	{16, 32}	32	{8, 16, 32}
Weight decay	{0, 0.1}	0	{0, 0.1}
Warmup steps	{0, 400}	0	{0, 1000}
Max sequence length	128	128	512
Peak learning rate	5e-5	5e-5	1e-5
Number of epochs	10		
Hidden dropout	0		
GeLU dropout	0		
Attention dropout	0		
Learning rate decay	Linear		
Optimizer	AdamW		
Adam ϵ	1e-6		
Adam (β_1, β_2)	(0.9, 0.999)		
Gradient clipping	0		

Table 3. Hyperparameters for fine-tuning on GLUE and SQuAD V1

C. Effectiveness of super pre-training.

We ask two questions towards evaluating the effectiveness of super pre-training: (a) Is the relative performance of sampled sub-networks on the MLM perplexity () correlated with performance of the same sub-networks when pre-trained individually from scratch ()?, and (b) Does the super pre-training afford sub-networks an advantage when being fine-tuned for tasks? For the first question, we sample a set of sub-networks T_S of both varying (33-96M) and similar (63-65M) parameter counts, and plot and in Figure 4 (b). We notice that and are highly correlated with a Spearman correlation coefficient of 0.954. This implies that the sub-network's measured MLM perplexity after super pre-training is a good proxy for final performance. We also observe that networks sampled at the similar parameter count (63-65M) have varying performance suggesting the sensitive role of shape in accuracy.

For studying the second question, we pre-train and then fine-tune the varying parameter count sub-networks (33-96M) in two ways (a) by retaining the weights learnt during super pre-training (), and (b) starting with random initialization . We plot two quantities in Figure 4 (c): the amount of pre-training time saved with (a) and the additional GLUE score obtained with (a). We observe that models with fewer parameters (30-50M) show significant savings in the pre-training time (up to $6.6\times$) and simultaneously benefit from improved GLUE accuracy (up to 3%). The gains on both axes for larger models are smaller. This suggests that smaller models whose parameters receive more weight updates due to sharing of the earlier rows and columns across sub-networks benefit more from super pre-training. This is encouraging because most effort in deployability is concerned with models of smaller size.

[1] https://huggingface.co/datasets/glue

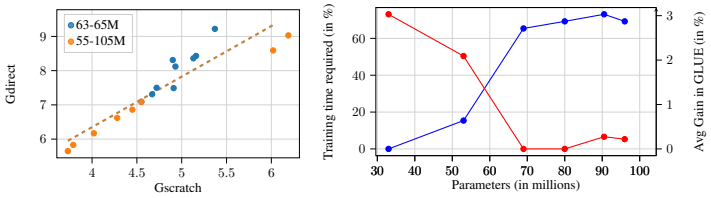

Figure 4. (a) SuperShaper is a fast and accurate proxy for sub-network perplexity, and (b) inherited sub-networks only require a fraction of pre-training cost (in blue) i.e. 1.3-6.6x reduction to reach optimum. This comes at a higher average gain in GLUE score (in red).

D. Efficient Deployment of SuperShaper sub-networks

Once the sub-networks are identified through evolutionary search or proposed heuristics, we combine the output bottleneck matrices of layer i with the input bottleneck matrices of layer $i+1$ for further parameter efficiency while retaining the functionality.

E. Feature importances for optimal sub-networks.

E.1. Perplexity Predictor importances.

F. Efficient Pytorch implementation

Pytorch code addition for slicing

```
class CustomLinear(nn.Linear):
    def __init__(
        self, super_in_dim,
            super_out_dim, bias=True,
            uniform_=None, non_linear="
            linear"
    ):
    self.samples = {}
    ...
    def set_sample_config(self,
        sample_in_dim, sample_out_dim):
       sample_weight = weight[:, :
           sample_in_dim]
       sample_weight = sample_weight[:
           sample_out_dim, :]
       self.samples["weight"] =
           sample_weight
       self.samples["bias"] = self.bias
           [..., : self.sample_out_dim]

    def forward(self, x):
       #override the Forward pass to use
            the sampled weights and bias
       return F.linear(x, self.samples["
           weight"], self.samples["bias"
           ])
```

The above code shows the additional lines added to PyTorch linear layer to support slicing for super pre-training. We add this to all the fundamental layers - *embedding layer*, *Linear layer* and *Layernorm* which adds up to 20 additional lines. This implementation is inspired from HAT[2]

G. Performance of SuperShaper

G.1. Fine-tuning T_S from SuperShaper

Table 6 compares the different methods of fine-tuning T_S, i.e. , , and respectively for a 63M network configuration obtained through evolutionary search. From the table, it is clear that and have better average GLUE performance. It is noteworthy, however, that SuperShaper is able to already provide good models that perform close to the best performance.

G.2. Comparing with BERT models

Table 4 shows the performance of a base model for SuperShaper, searched for 100M constraint compared against BERT-Base. As discussed in the main paper, SuperShaper provides models that match the performance of BERT-Base models for a significantly fewer parameters.

G.3. Average GLUE performance of best models from Evolutionary Search

Table 5 shows the average GLUE performance for all the best models found through evolutionary search for reference.

G.4. Comparison with HAT(Wang et al., 2020a) and OFA(Cai et al., 2019)

HAT compresses encoder-decoder models by elasticizing number of layers, hidden size and number of attention heads for machine translation task while OFA proposed a compression for CNN based models on image classification task. To compare our approach with these techniques, We use the results from (Zhang et al., 2021) who reimplement these approaches and report on three Glue tasks - MRPC, SST2 and RTE for 2 compression ratios - 0.75x and 0.5x. We compare these results against our pareto evolutionary search models that have compression ratios of 0.78x (90.5M) and 0.54x (63M). The results are reported in table Table 8 and

[2]https://github.com/mit-han-lab/hardware-aware-transformers

Model	Params	MNLI-m	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	Avg. GLUE	Squad V1
BERT-Base (from NAS-BERT)	110M	85.2	91	91.3	61	92.9	90.3	76	87.7	84.4	81.8 / 88.9
BERT-Base (from DistilBERT)	110M	86.7	89.6	91.8	56.3	92.7	89	69.3	88.6	83	81.2 / 88.5
SuperShaper (ours)	96M	83.9	90.86	90.92	56.58	92.89	88.3	77.98	88.48	83.7	80.19 / 88.2

Table 4. Comparing SuperShaper with BERT-Base models.

Params (M)			MNLI-m	QQP	QNLI	CoLA	SST-2	STS-B	RTE	MRPC	Average GLUE
33	10.82	12.44	73.45	84.71	80.52	10.27	85.32	82.65	65.70	82.11	70.59
53	8.59	6.02	79.40	89.51	86.38	33.85	89.11	86.66	68.23	84.56	77.21
63	7.09	4.55	82.23	90.18	88.05	53.00	91.86	87.63	79.06	89.46	82.68
69	6.62	4.28	82.74	90.45	89.54	54.98	91.28	88.42	77.98	87.75	82.89
80	6.17	4.02	83.05	90.56	89.22	54.87	93.10	88.46	80.14	87.75	83.39
90.5	5.83	3.79	83.06	90.51	88.72	58.87	91.51	88.47	77.26	88.97	83.42
96	5.65	3.73	83.90	90.86	90.92	56.58	92.89	88.30	77.98	88.48	83.74

Table 5. Performance of best models from parameter-constrained evolutionary search

Shapes	Params (M)	G_direct	G_scratch	L1	L2	L3	L4	L5	L6	L7	L8	L9	L10	L11	L12
EvoSearch 1	65	6.86	4.45	480	360	360	240	240	360	480	480	360	480	540	540
Evo Search 2	63	7.09	4.55	480	240	360	240	540	480	360	360	360	360	540	480
Lower Triangle	64	7.31	4.67	120	120	240	240	360	360	360	480	540	540	600	768
Random	64	7.49	4.91	480	360	360	540	480	540	360	480	540	120	360	540
Rectangle	58	7.5	4.72	360	360	360	360	360	360	360	360	360	360	360	360
Inverted Diamond	65	8.12	4.93	768	600	360	240	240	120	120	240	240	360	600	768
Bottle	64	8.31	4.9	120	120	120	120	120	120	600	600	600	600	600	768
Diamond	64	8.36	5.13	120	240	360	480	480	540	768	540	480	360	240	120
Upper Triangle	64	8.43	5.16	768	600	540	540	480	360	360	360	240	240	120	120
Inverted Bottle	64	9.22	5.37	768	600	600	600	600	600	120	120	120	120	120	120

Table 6. Hidden dimensions of templatized shapes and their corresponding perplexities for and .

Evo-search parameters		Parameter range	Number of networks	Spearman Correlation	Pearson Correlation
53	8.59	52-54	54	71.03	67.95
63	7.09	62-64	704	80.34	82.52
65	6.86	63-65	862	69.47	71.34
69	6.62	68-70	1065	47.22	52.06
80	6.17	79-81	486	72.32	67.34
90.5	5.83	89-91	47	56.8	54.67
96	5.65	95-97	6	65.71	69.65

Table 7. Shape difference positively correlates with difference across a wide parameter range

we see that SuperShaper outperforms both these approaches with a significant margin.

Model	MRPC		SST2		RTE		AVG
	Compression Ratio						
	0.75x	**0.5x**	**0.75x**	**0.5x**	**0.75x**	**0.5x**	
HAT-BERT	82.2	82.6	88.6	88.6	65.0	64.6	78.6
OFA-BERT	87.6	85.2	89.3	89.8	62.8	65.3	80.0
YOCO-BERT	90.4	87.6	92.9	91.9	75.1	69.3	84.5
SuperShaper(ours)$*$	88.97	89.46	91.51	91.86	77.26	79.06	**86.4**

Table 8. Comparison with HAT, OFA and YocoBert with SuperShaper. $*$ We use models with compression ratios of 0.78x (90.5M) and 0.54x (63M)