GhostBERT: Generate More Features with Cheap Operations for BERT

Zhiqi Huang[1], **Lu Hou**[2], **Lifeng Shang**[2], **Xin Jiang**[2], **Xiao Chen**[2], **Qun Liu**[2]
[1]Peking University, [2]Huawei Noah's Ark Lab
zhiqihuang@pku.edu.cn, {houlu3, shang.lifeng, jiang.xin, chen.xiao, qun.liu}@huawei.com

Abstract

Transformer-based pre-trained language models like BERT, though powerful in many tasks, are expensive in both memory and computation, due to their large number of parameters. Previous works show that some parameters in these models can be pruned away without severe accuracy drop. However, these redundant features contribute to a comprehensive understanding of the training data and removing them weakens the model's representation ability. In this paper, we propose GhostBERT, which generates more features with very cheap operations from the remaining features. In this way, GhostBERT has similar memory and computational cost as the pruned model, but enjoys much larger representation power. The proposed ghost module can also be applied to unpruned BERT models to enhance their performance with negligible additional parameters and computation. Empirical results on the GLUE benchmark on three backbone models (i.e., BERT, RoBERTa and ELECTRA) verify the efficacy of our proposed method.

1 Introduction

Recently, there is a surge of research interests in compressing the transformer-based pre-trained language models like BERT into smaller ones using various compression methods, i.e., knowledge distillation (Sanh et al., 2019; Sun et al., 2019; Jiao et al., 2020), pruning (Michel et al., 2019; Fan et al., 2019), low-rank approximation (Lan et al., 2020), weight-sharing (Lan et al., 2020), dynamic networks with adaptive depth and/or width (Liu et al., 2020; Hou et al., 2020; Xin et al., 2020; Zhou et al., 2020), and quantization (Shen et al., 2020; Fan et al., 2020; Zhang et al., 2020; Bai et al., 2021).

Previous works show that there are some redundant features in the BERT model, and unimportant attention heads or neurons can be pruned away

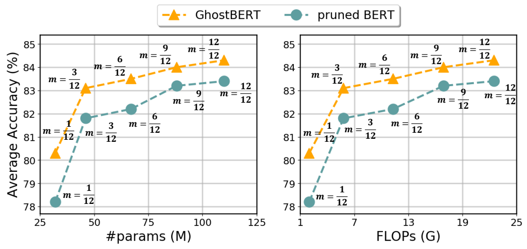
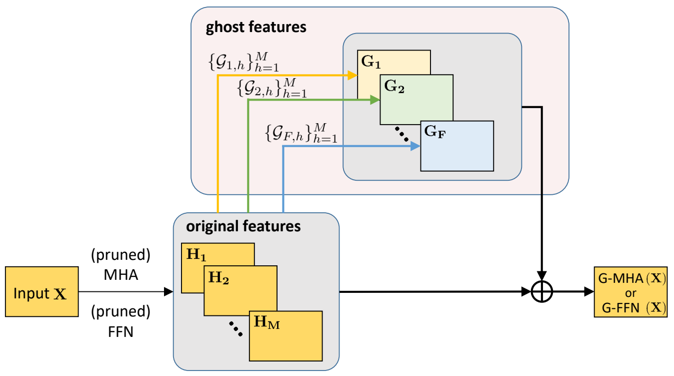


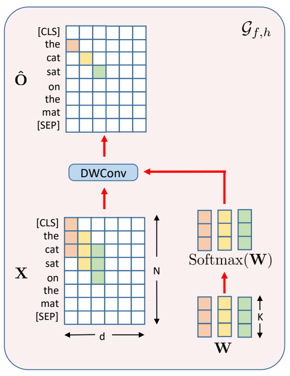
Figure 1: Average GLUE development accuracy versus #params and FLOPs with the (pruned) BERT and our GhostBERT. m is the width multiplier of the model.

without severe accuracy degradation (Michel et al., 2019; Hou et al., 2020). However, for computer vision (CV) tasks, it is shown in (Han et al., 2020) that redundant features in convolutional neural networks also contribute positively to the performance, and using cheap linear operations to generate more ghost feature maps enhances the performance with few additional parameters. On the other hand, it is shown in (Voita et al., 2019; Kovaleva et al., 2019; Rogers et al., 2020) that many attention maps in pre-trained language models exhibit typical positional patterns, e.g., diagonal or vertical, which can be easily generated from other similar ones using operations like convolution.

Based on the above two aspects, in this paper, we propose to use cheap ghost modules on top of the remaining important attention heads and neurons to generate more features, so as to compensate for the pruned ones. Considering that the convolution operation (1) encodes local context dependency, as a complement of the global self-attention in Transformer models (Wu et al., 2020); and (2) can generate some BERT features like positional attention maps from similar others, in this work, we propose to use the efficient 1-Dimensional Depthwise Separable Convolution (Wu et al., 2019) as the basic operation in the ghost module. To ensure the generated ghost features have similar scales



(a) Adding ghost modules $\{\mathcal{G}_{f,h}\}_{f=1,h=1}^{F,M}$ to MHA and FFN.



(b) Ghost Module $\mathcal{G}_{f,h}$.

Figure 2: Using ghost modules to generate more features in BERT. G-MHA/FFN stands for Ghost-MHA/FFN.

as the original ones, we use a softmax function to normalize the convolution kernel.

Afterwards, we fine-tune the parameters in both the BERT backbone model and the added ghost modules. Note that the ghost modules are not necessarily applied to pruned models. They can also be directly applied to pre-trained language models for better performance while with negligible additional parameters and floating-point operations (FLOPs). Figure 1 summarizes the average accuracy versus parameter size and FLOPs on the GLUE benchmark, where adding ghost modules to both the unpruned ($m = 12/12$) and pruned ($m < 1$) BERT models perform better than the counterparts without ghost modules. More experiments on the GLUE benchmark show that with only 0.4% more parameters and 0.9% more FLOPs, the proposed ghost modules improve the average accuracy of BERT-base, RoBERTa-base and ELECTRA-small by $0.9, 0.6, 2.4$ points, respectively. When applying ghost modules to small or pruned models, the resultant models outperform other BERT compression methods.

2 Approach

In this section, we first introduce where to add ghost modules in a BERT model (Section 2.1), and then discuss the components and optimization details of the ghost module (Section 2.2).

2.1 Adding Ghost Modules to BERT

The BERT model is built with Transformer layers, each of which contains a Multi-Head Attention (MHA) layer and a Feed-Forward Network (FFN), as well as skip connections and layer normalizations. Hou et al. (2020) show that the computations for attention heads of MHA and neurons in the intermediate layer of FFN can be performed in parallel. Thus the BERT model can be compressed in a structured manner by pruning parameters associated with these heads and neurons (Hou et al., 2020). In this paper, after pruning the unimportant heads and neurons, we employ cheap ghost modules upon the remaining ones to generate more ghost features to compensate for the pruned ones.

For simplicity of notation, we omit the bias terms in linear and convolution operations where applicable in the rest of this work.

2.1.1 Ghost Module on MHA

Following (Hou et al., 2020), we divide the computation of MHA into the computation of each attention head. Specifically, suppose the sequence length and hidden state size are n and d, respectively. Each transformer layer consists of N_H attention heads. For input matrix $\mathbf{X} \in \mathbb{R}^{n\times d}$, the h-th attention head computes its output as $\mathbf{H}_h(\mathbf{X}) = \text{Softmax}(1/\sqrt{d} \cdot \mathbf{X}\mathbf{W}_h^Q\mathbf{W}_h^{K\top}\mathbf{X}^\top)\mathbf{X}\mathbf{W}_h^V\mathbf{W}_h^{O\top}$, where $\mathbf{W}_h^Q, \mathbf{W}_h^K, \mathbf{W}_h^V, \mathbf{W}_h^O \in \mathbb{R}^{d\times d_h}$ with $d_h = d/N_H$ are the projection matrices associated with it. In multi-head attention, N_H heads are computed in parallel to get the final output:

$$\text{MHA}(\mathbf{X}) = \sum_{h=1}^{N_H} \mathbf{H}_h(\mathbf{X}). \tag{1}$$

Given a width multiplier $m \leq 1$, we keep $M = \lfloor N_h m \rfloor$ heads and use them to generate F ghost features. The f^{th} ghost feature is generated by

$$\mathbf{G}_f(\mathbf{X}) = \text{Nonlinear}\left(\sum_{h=1}^{M} \mathcal{G}_{f,h}\left(\mathbf{H}_h(\mathbf{X})\right)\right). \tag{2}$$

where $\mathcal{G}_{f,h}$ is the proposed cheap ghost module which generates features from the h^{th} attention head's representation to the f^{th} ghost feature. ReLU is used as the nonlinearity function. Thus the computation of MHA in the GhostBERT is:

$$\text{Ghost-MHA}(\mathbf{X}) = \sum_{h=1}^{M} \mathbf{H}_h(\mathbf{X}) + \sum_{f=1}^{F} \mathbf{G}_f(\mathbf{X}). \quad (3)$$

Besides being added to the output of MHA, the ghost modules can also be added to other positions in MHA. Detailed discussions are in Section 4.2.

2.1.2 Ghost Module on FFN

Similar to the attention heads in MHA, the computation of FFN can also be divided into computations for each neuron in the intermediate layer of FFN (Hou et al., 2020). With a slight abuse of notation, we still use $\mathbf{X} \in \mathbb{R}^{n \times d}$ as the input to FFN. Denote the number of neurons in the intermediate layer as d_{ff}, the computation of FFN can be written as: $\text{FFN}(\mathbf{X}) = \sum_{i=1}^{d_{ff}} \text{GeLU}\left(\mathbf{X}\mathbf{W}^1_{:,i}\right) \mathbf{W}^2_{i,:}$, where $\mathbf{W}^1, \mathbf{W}^2$ are the weights in FFN.

For simplicity, we also use width multiplier m for FFN as MHA, and divide these neurons into N_H folds, where each fold contains $d_f = d_{ff}/N_H$ neurons. For the h-th fold, its output can be computed as $\mathbf{H}_h(\mathbf{X}) = \text{GeLU}(\mathbf{X}\mathbf{W}^1_h) \mathbf{W}^2_h$ where $\mathbf{W}^1_h = \mathbf{W}^1_{:,(h-1)d_f:hd_f}$ and $\mathbf{W}^2_h = \mathbf{W}^2_{(h-1)d_f:hd_f,:}$ are the parameters associated with it. In FFN, N_H folds are computed in parallel to get the output:

$$\text{FFN}(\mathbf{X}) = \sum_{h=1}^{N_H} \mathbf{H}_h(\mathbf{X}). \quad (4)$$

For width multiplier m, we keep M folds of neurons and use ghost modules to generate F ghost features as in Equation (2). Thus the computation of FFN in the GhostBERT can be written as:

$$\text{Ghost-FFN}(\mathbf{X}) = \sum_{h=1}^{M} \mathbf{H}_h(\mathbf{X}) + \sum_{f=1}^{F} \mathbf{G}_f(\mathbf{X}). \quad (5)$$

2.2 Ghost Module

In the previous section, we discussed where we insert the ghost modules in the Transformer layer. In this section, we elaborate on the components and normalization of the ghost modules.

Generally speaking, any function can be used as the ghost module $\mathcal{G}$ in Equation (2). Considering that (i) convolution operation can encode local context dependency, as a compensation for the global self-attention (Wu et al., 2020; Jiang et al., 2020); and (ii) features like diagonal or vertical attention maps (Kovaleva et al., 2019; Rogers et al., 2020) can be easily generated by convolving similar others, we consider using convolution as the basic operation in the ghost module.

2.2.1 Convolution Type

With a slight abuse of notation, here we still use $\mathbf{X} \in \mathbb{R}^{n \times d}$ as the input to the convolution, i.e., the output $\mathbf{H}_h$ of h^{th} head in MHA or h^{th} fold of neurons in FFN. Denote $\mathbf{O} \in \mathbb{R}^{n \times d}$ as the output of the convolution in the ghost module.

1-Dimensional convolution (Conv1D) over the sequence direction encodes local dependency over contexts, and has shown remarkable performance for NLP tasks (Wu et al., 2019, 2020). To utilize the representation power of Conv1D without too much additional memory and computation, we choose 1-Dimensional Depthwise Separable Convolution (DWConv) (Wu et al., 2019) for the ghost module. Compared with Conv1D, DWConv performs a convolution independently over every channel, and reduces the number of parameters from d^2k to dk (where k is the convolution kernel size). Denote the weight of the DWConv operation as $\mathbf{W} \in \mathbb{R}^{d \times k}$. After applying DWConv, the output for the i^{th} token and c^{th} channel can be written as:

$$\begin{aligned} O_{i,c} &= \text{DWConv}(\mathbf{X}_{:,c}, \mathbf{W}_{c,:}, i, c) \\ &= \sum_{m=1}^{k} W_{c,m} \cdot X_{i-\lceil \frac{k+1}{2} \rceil + m, c}. \end{aligned}$$

2.2.2 Normalization

Since the parameters of the BERT backbone model and the ghost modules can have quite different scales and optimization behaviors, we use a softmax function to normalize each convolution kernel $\mathbf{W}_{c,:}$ across the sequence dimension as $\text{Softmax}(\mathbf{W}_{c,:})$ before convolution as Wu et al. (2019). By softmax normalization, the weights in one kernel are summed up to 1, ensuring that the convolved output has a similar scale as the input. Thus after applying the ghost module, the output for the i^{th} token and c^{th} channel can be written as:

$$\hat{O}_{i,c} = \text{DWConv}(\mathbf{X}_{:,c}, \text{Softmax}(\mathbf{W}_{c,:}), i, c).$$

2.3 Training Details

To turn a pre-trained BERT model into a smaller-sized GhostBERT, we do the following three steps:

Model-Size	FLOPs(G)	#params(M)	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
BERT-base (Devlin et al., 2019)	22.5	110	84.5	92.0	90.9	71.1	92.9	87.8	58.1	89.8	83.4
GhostBERT (m = 12/12)	22.5	110	84.7	92.3	91.1	71.8	93.0	88.0	63.6	89.7	**84.3**
GhostBERT (m = 9/12)	16.9	88	84.8	92.1	91.2	72.6	92.6	87.5	61.1	89.8	84.0
GhostBERT (m = 6/12)	11.3	67	84.7	92.2	91.2	72.2	92.9	87.3	58.1	89.2	83.5
GhostBERT (m = 3/12)	5.8	46	84.3	91.6	91.4	72.9	94.6	86.5	53.9	89.2	83.1
GhostBERT (m = 1/12)	2.0	32	82.8	90.0	90.5	66.1	92.8	86.0	46.1	87.8	80.3
RoBERTa-base (Liu et al., 2019)	22.5	125	87.6	92.8	91.9	78.7	94.8	90.2	63.6	91.2	86.4
GhostRoBERTa (m = 12/12)	22.5	125	88.0	93.1	91.9	80.5	95.3	90.7	65.0	91.3	**87.0**
GhostRoBERTa (m = 9/12)	16.9	103	87.6	92.9	91.9	79.4	95.4	89.0	60.8	90.7	86.0
GhostRoBERTa (m = 6/12)	11.3	82	86.8	92.6	91.6	77.6	94.4	89.7	57.6	90.3	85.1
GhostRoBERTa (m = 3/12)	5.8	61	86.1	91.7	91.2	73.6	94.5	88.0	52.4	89.2	83.3
GhostRoBERTa (m = 1/12)	2.0	47	82.1	89.2	90.5	66.1	93.7	83.3	39.8	87.4	79.0
ELECTRA-small (Clark et al., 2020)	1.7	14	78.9	87.9	88.3	68.5	88.3	87.4	56.8	86.8	80.4
GhostELECTRA-small (m = 4/4)	1.7	14	82.5	89.3	90.7	71.5	92.0	88.7	59.6	88.4	**82.8**

Table 1: Development set results of the baseline pre-trained language models and our proposed method on the GLUE benchmark. Both pruned and unpruned BERT-base (resp. RoBERTa-base) are used as the backbone models for GhostBERT (resp. GhostRoBERTa). The unpruned ELECTRA-small is used as the backbone model for the GhostELECTRA-small. m is the width multiplier written in the form of proportion, whose numerator and denominator represent the remaining attention heads/folds of neurons and the total number of heads/folds, respectively.

Pruning. For a certain width multiplier m, we prune the attention heads in MHA and neurons in the intermediate layer of FFN from a pre-trained BERT-based model following (Hou et al., 2020).

Distillation. Then we add ghost modules to the pruned model as in Section 2.1. Suppose there are L Transformer layers. We distill the knowledge from the embedding (i.e., the output of the embedding layer) $\mathbf{E}$, hidden states $\mathbf{M}_l$ after MHA and $\mathbf{F}_l$ after FFN (where $l = 1, 2, \cdots, L$) from the full-sized teacher model to $\mathbf{E}^m, \mathbf{M}_l^m, \mathbf{F}_l^m$ of the student GhostBERT. Following (Jiao et al., 2020), we use the augmented data for distillation. Denote MSE as the mean squared error, the three loss terms are $\ell_{emb} = \text{MSE}(\mathbf{E}^m, \mathbf{E})$, $\ell_{mha} = \sum_{l=1}^{L} \text{MSE}(\mathbf{M}_l^m, \mathbf{M}_l)$, and $\ell_{ffn} = \sum_{l=1}^{L} \text{MSE}(\mathbf{F}_l^m, \mathbf{F}_l)$, respectively. Thus, the distillation loss function is:

$$\mathcal{L}_{distill} = \ell_{emb} + \ell_{mha} + \ell_{ffn}.$$

Fine-tuning. Denote $\mathbf{y}$ as the predicted logits, we finally fine-tune the GhostBERT with ground-truth labels $\hat{\mathbf{y}}$ as:

$$\mathcal{L}_{finetune} = \text{CrossEntropy}(\hat{\mathbf{y}}, \mathbf{y}).$$

Note that instead of being applied to pruned models, the cheap ghost modules can also be directly applied to a pre-trained model for better performance while with negligible additional parameters and FLOPs. In this case, the training procedure contains only the distillation and fine-tuning steps.

Empirically, to save memory and computation, we generate one ghost feature for each MHA or FFN (i.e., $F = 1$ in Equations (3) and (5)), and let all ghost modules $\mathcal{G}_{f,h}$ share the same parameters with each other. As will be shown in Section 3, adding these simplified ghost modules already achieve clear performance gain empirically.

3 Experiment

In this section, we show the efficacy of the proposed method with (pruned) BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019) and ELECTRA (Clark et al., 2020) as backbone models.

3.1 Setup

Experiments are performed on the GLUE benchmark (Wang et al., 2019), which consists of various natural language understanding tasks. More statistics about the GLUE datasets are in Appendix A.1. Following (Clark et al., 2020), we report Spearman correlation for STS-B, Matthews correlation for CoLA and accuracy for the other tasks. For MNLI, we report the results on the matched section. The convolution kernel size in the ghost module is set as 3 unless otherwise stated. The detailed hyperparameters for training the GhostBERT are in Appendix A.2. The model with the best development set performance is used for testing. For each method, we also report the number of parameters

Model	FLOPs(G)	#params(M)	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
BERT-base (Devlin et al., 2019)	22.5	110	84.6	90.5	89.2	66.4	93.5	84.8	52.1	85.8	80.9
RoBERTa-base (Liu et al., 2019)	22.5	125	86.0	92.5	88.7	73.0	94.6	86.5	50.5	88.1	82.5
ELECTRA-small (Clark et al., 2020)	1.7	14	79.7	87.7	88.0	60.8	89.1	83.7	54.6	80.3	78.0
TinyBERT$_6$ (Jiao et al., 2020)	11.3	67	84.6	90.4	89.1	70.0	93.1	87.3	51.1	83.7	81.2
TinyBERT$_4$ (Jiao et al., 2020)	1.2	15	82.5	87.7	89.2	66.6	92.6	86.4	44.1	80.4	78.7
ConvBERT-medium (Jiang et al., 2020)	4.7	17	82.1	88.7	88.4	65.3	89.2	84.6	56.4	82.9	79.7
ConvBERT-small (Jiang et al., 2020)	2.0	14	81.5	88.5	88.0	62.2	89.2	83.3	54.8	83.4	78.9
MobileBERT w/o OPT (Sun et al., 2020)	5.7	25	84.3	91.6	88.3	70.4	92.6	84.5	51.1	84.8	81.0
MobileBERT (Sun et al., 2020)	5.7	25	83.3	90.6	-	66.2	92.8	-	50.5	84.4	-
MobileBERT-tiny (Sun et al., 2020)	3.1	15	81.5	89.5	-	65.1	91.7	-	46.7	80.1	-
GhostBERT (m = 12/12)	22.5	110	84.6	91.1	89.3	70.2	93.1	86.9	54.6	83.8	81.7
GhostBERT (m = 9/12)	16.9	88	84.9	91.0	88.6	69.2	92.9	86.1	53.7	84.0	81.3
GhostBERT (m = 6/12)	11.3	67	84.2	90.8	89.1	69.6	93.1	84.0	53.4	83.1	80.9
GhostBERT (m = 3/12)	5.8	46	83.8	90.7	89	68.6	93.2	82.5	51.3	82.5	80.2
GhostBERT (m = 1/12)	2.0	32	82.5	89.3	88.7	65.0	92.9	81.0	41.3	80.0	77.6
GhostRoBERTa (m = 12/12)	22.5	125	87.9	93.0	89.6	74.6	95.1	88.0	52.4	88.3	83.6
GhostRoBERTa (m = 9/12)	16.9	103	87.7	92.6	89.5	73.0	94.5	85.7	51.9	87.1	82.8
GhostRoBERTa (m = 6/12)	11.3	82	86.3	92.1	89.5	71.5	94.5	86.8	51.2	87.0	82.4
GhostRoBERTa (m = 3/12)	5.8	61	85.5	91.2	89.1	68.5	93.4	85.3	48.9	84.7	80.8
GhostRoBERTa (m = 1/12)	2.0	47	81.3	88.6	88.5	62.8	92.1	82.8	39.7	81.8	77.2
GhostELECTRA-small (m = 4/4)	1.7	14	82.3	88.3	88.5	64.7	91.9	88.4	55.8	83.5	80.4

Table 2: Test set results of the baseline pre-trained language models, BERT compression methods and our proposed method on the GLUE benchmark.

and FLOPs at inference (Details can be found in Appendix A.3).

We compare our proposed method against the following methods: (i) baseline pre-trained language models: BERT-base (Devlin et al., 2019), RoBERTa-base (Liu et al., 2019) and ELECTRA-small (Clark et al., 2020); (ii) BERT compression methods: TinyBERT (Jiao et al., 2020), ConvBERT (Jiang et al., 2020), and MobileBERT (Sun et al., 2020). The development set results of RoBERTa-base are from Hou et al. (2020). The test set results of ELECTRA, BERT-base and ConvBERT are from Jiang et al. (2020). The others are from their original papers or repositories.

3.2 Main Results

3.2.1 Ghost Modules on Unpruned Models

Table 1 shows the GLUE development set results of the baseline pre-trained language models and our proposed method. When the cheap ghost modules are directly applied to these unpruned pre-trained models, better performances are achieved with only negligible additional parameters and FLOPs. Specifically, adding ghost modules to BERT-base, RoBERTa-base and ELECTRA-small increases the average development accuracy by $0.9, 0.6, 2.4$ points with only 55.3K more parameters, and 14.2M more FLOPs. For the test set, the average performance gains are $0.8, 1.1, 2.4$ points.

3.2.2 Ghost Modules on Pruned Models

Comparison with Baseline Models. From Table 1, when the ghost modules are applied to the pruned BERT (or RoBERTa) model with $m < 1$, the proposed GhostBERT or GhostRoBERTa also achieves comparable performances as BERT-base or RoBERTa-base with fewer FLOPs. Specifically, GhostBERT (m = 6/12) and GhostRoBERTa (m = 9/12) perform similarly or even better than BERT-base and RoBERTa-base with only 50% and 75% FLOPs, respectively. In particular, when the compression ratio increases (i.e., m = 3/12, 1/12), we still achieve 99.6% performance (resp. 96.3%) with only 25% FLOPs (resp. 8%) of BERT-base model.

Comparison with Other Compression Methods. Table 2 shows the comparison between the proposed method and other popular BERT compression methods. Under similar parameter sizes or FLOPs, the proposed GhostBERT performs comparably as the other BERT compression methods, while GhostRoBERTa often outperforms them. In particular, GhostELECTRA-small has over 1.5 points or higher accuracy gain than other similar-sized small models like ELECTRA-small, TinyBERT$_4$ and ConvBERT-small.

In Table 3 and Figure 1, we also compare the pruned BERT with and without ghost modules. For fair comparison, for the pruned model without ghost module, we use the same training procedure as Section 2.3. As can be seen, adding the ghost modules achieves considerable improvement with negligible additional memory and computation.

m	Pruned BERT			GhostBERT		
	FLOPs (G)	#params (M)	Avg. acc	FLOPs (G)	#params (M)	Avg. acc
1/12	2.0	32	78.2	2.0	32	**80.3**
3/12	5.8	46	81.8	5.8	46	**83.1**
6/12	11.3	67	82.2	11.3	67	**83.5**
9/12	16.9	88	83.2	16.9	88	**84.0**

Table 3: Comparison of GhostBERT and pruned BERT. m stands for the width multiplier.

3.3 Ablation Study

In this section, we perform ablation study in the (i) training procedure: including data augmentation (DA) and knowledge distillation (KD); (ii) ghost module: including convolution kernel size, softmax normalization over the convolution kernel and nonlinearity for each ghost feature in Equation (2).

Training Procedure. Table 4 verifies the effectiveness of the Data Augmentation (DA) and Knowledge Distillation (KD) upon the GhostBERT model with width multiplier $m \in \{3/12, 1/12\}$. The GhostBERT incurs severe accuracy drop without DA and KD. with a drop of 3.5 and 6.4 points on average, for $m = 3/12$ and $1/12$, respectively.

Ghost Module. Table 4 also shows the effectiveness of the softmax normalization over the convolution kernel and ReLU nonlinearity in Equation (2). As can be seen, dropping the softmax normalization or ReLU nonlinearity reduces the average accuracy by 0.8 and 1.6 points respectively for $m = 3/12$, and 0.9 and 2.2 points respectively for $m = 1/12$.

Further, we explore whether the kernel size plays an important role in the DWConv in the ghost module. Figure 3 shows the results of GhostBERT with width multipliers $m \in \{3/12, 1/12\}$, with various convolution kernel sizes in DWConv. Average accuracy over five tasks is reported. Detailed results for each task can be found in Table 9 in Appendix B.1. As can be seen, the performance of GhostBERT increases first and then decreases gradually as the kernel size increases. For both width multipliers, kernel size 3 performs best and is used as the default kernel size in other experiments unless otherwise stated.

4 Discussion

In this section, we discuss about different choices of which type of convolution to use in the ghost module (Section 4.1), and where to posit the ghost modules in a BERT model (Section 4.2).

4.1 Ghost Module Types

Besides the DWConv in Section 2.2, in this section, we discuss more options for the convolution in the ghost module. We follow the notation in Section 2.2 and denote the input, output, kernel size of the convolution as $\mathbf{X}$, $\mathbf{W}$ and k, respectively.

1-Dimensional Convolution. If the kernel convolves input over the sequence direction (abbreviated as Conv1D_S), the number of input and output channel is d, and the weight $\mathbf{W}$ has shape $\mathbf{W} \in \mathbb{R}^{d\times d\times k}$. After applying Conv1D_S, the output for the i^{th} token and c^{th} channel is:

$$\begin{aligned} O_{i,c} &= \text{Conv1D\_S}(\mathbf{X}, \mathbf{W}_{c,:,:}, i, c) \\ &= \sum_{j=1}^{d}\sum_{m=1}^{k} W_{c,j,k} \cdot X_{i-\lceil\frac{k+1}{2}\rceil+m,j}. \end{aligned}$$

If the kernel convolves input over the feature direction (abbreviated as Conv1D_F), the number of input and output channel is n, and the weight has shape $\mathbf{W} \in \mathbb{R}^{n\times n\times k}$. After applying Conv1D_F, the output for the i^{th} token and c^{th} channel is:

$$\begin{aligned} O_{i,c} &= \text{Conv1D\_F}(\mathbf{X}, \mathbf{W}_{i,:,:}, i, c) \\ &= \sum_{j=1}^{n}\sum_{m=1}^{k} W_{i,j,m} \cdot X_{j,c-\lceil\frac{k+1}{2}\rceil+m}. \end{aligned}$$

2-Dimensional Convolution (Conv2D). For Conv2D, the number of input and output channels are both 1, and thus the weight $\mathbf{W}$ has shape $\mathbf{W} \in \mathbb{R}^{1\times 1\times k\times k}$. After applying Conv2D, the output for the i^{th} token and c^{th} channel is:

$$\begin{aligned} O_{i,c} &= \text{Conv2D}(\mathbf{X}, \mathbf{W}, i, c) \\ &= \sum_{w=1}^{k}\sum_{h=1}^{k} W_{:,:,h,w} \cdot X_{i-\lceil\frac{k+1}{2}\rceil+h,c-\lceil\frac{k+1}{2}\rceil+w}. \end{aligned}$$

4.1.1 Comparison of Different Convolutions

Table 5 shows the comparison of using different convolutions for the ghost module. For 1-Dimensional convolution, Conv1D_S performs better Conv1D_F. This may because that convolving over the sequence urges the model to learn the dependencies among tokens.

m	Model	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
3/12	GhostBERT	84.3	91.6	91.4	72.9	94.6	86.5	53.9	89.2	**83.1**
	- DA & KD	80.2	88.5	90.0	63.2	91.6	83.8	52.5	86.7	79.6
	- Softmax	84.3	91.5	90.9	71.8	92.3	85.5	52.6	89.1	82.3
	- ReLU	84.0	91.7	91.0	70.8	92.3	85.8	47.6	88.6	81.5
1/12	GhostBERT	82.8	90.0	90.5	66.1	92.8	86.0	46.1	87.8	**80.3**
	- DA & KD	76.0	83.4	86.6	58.1	86.6	80.6	35.8	84.4	73.9
	- Softmax	82.6	90.0	90.4	65.3	92.1	85.5	40.8	88.1	79.4
	- ReLU	82.7	89.8	90.6	60.3	92.0	84.8	37.8	87.1	78.1

Table 4: Ablation study of data augmentation (DA), knowledge distillation (KD), softmax normalization over the convolution kernel, and non-linearity. Results on the GLUE development set are reported.

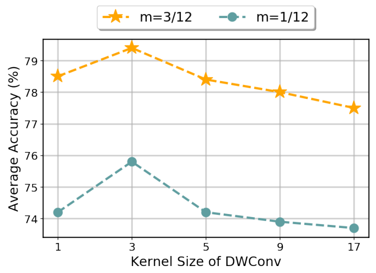


Figure 3: Average score over five tasks with various kernel sizes of DWConv in the ghost module.

m	Convolution Type	FLOPs(G)	#params(M)	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
3/12	Conv1D_S	16.6	88	83.6	91.3	91.1	70.0	92.5	86.5	52.3	89.3	82.1
	Conv1D_F	7.6	47	84.0	91.5	91.0	61.7	92.2	87.0	48.9	89.0	80.7
	Conv2D	5.8	46	83.7	91.7	91.0	68.2	92.5	86.5	50.4	89.3	81.7
	ours: DWConv	5.8	46	84.3	91.6	91.4	72.9	94.6	86.5	53.9	89.2	**83.1**
1/12	Conv1D_S	12.9	74	82.6	90.1	90.4	66.1	92.0	85.5	47.5	88.2	**80.3**
	Conv1D_F	3.9	33	82.2	86.6	90.0	54.9	92.3	72.8	31.2	87.3	74.7
	Conv2D	2.1	32	81.7	89.2	90.1	63.2	91.7	83.8	37.9	88.0	78.2
	ours: DWConv	2.0	32	82.8	90.0	90.5	66.1	92.8	86.0	46.1	87.8	**80.3**

Table 5: Comparison of different convolutions in the ghost module. The convolution kernel size is 3. The backbone model is BERT-base. Results on the GLUE development set are reported.

Though 2-Dimensional convolution (Conv2D) is quite successful in CV tasks, it performs much worse than Conv1D_S here. This may because the two dimensions of feature maps in CV tasks encode similar information, while those of hidden states in Transformers encode quite different information (i.e., feature and sequence). Thus Conv2D results in worse performance than Conv1D_S, though much fewer parameters and FLOPs are required.

On the other hand, DWConv achieves comparable performance as Conv1D_S, while being much more efficient in terms of number of parameters and FLOPs, by performing the convolution independently over every feature dimension.

4.2 Ghost Module Positions

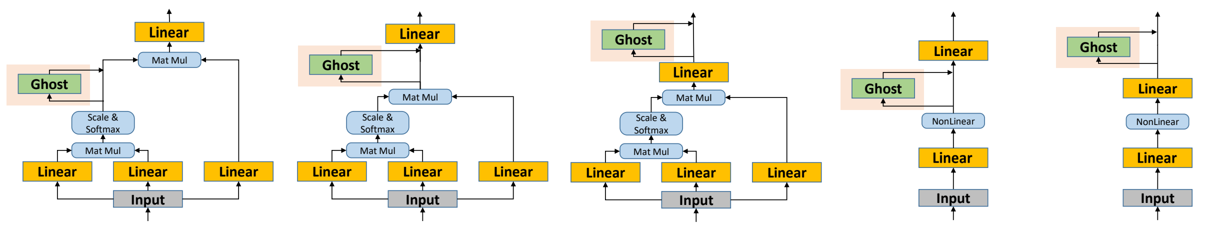
In this section, we explore more possible positions of adding the ghost module. For MHA, besides adding ghost module after the projection layer (`After O` in Figure 4(c)) as in Section 2.1.1, we can also add it right after calculating the attention score (`After QK` in Figure 4(a)), or after multiplying the attention score and the value layer (`After V` in Figure 4(b)). For FFN, besides adding the ghost module after the second linear layer (`After FFN2` in Figure 4(e)) as in Section 2.1.1, we can also add it after the intermediate layer (`After FFN1` in Figure 4(d)). Note that we use Conv2D as the ghost module for `After QK` because the attention map encodes attention probabilities in both dimensions. For `After QK` and `After V`, to match the dimension of other parameters, the number of input and output channels are M and $N_H - M$, respectively.

Table 6 shows the results of adding one ghost module to the same position for each Transformer layer. As can be seen, adding ghost module upon the attention maps (`After QK`) performs best. However, since the parameters in the value and projection layer of MHA are left unpruned, `After QK` has much more parameters and FLOPs than the other positions. Adding ghost modules to the other four positions has similar average accuracy. Thus in this work, for MHA, we choose the most memory- and computation-efficient strategy `After O`. Similarly, for FFN, we also add ghost modules to the final output (`After FFN2`). From Table 6, our way of adding ghost modules has comparable performance as `After QK`, while being much more efficient in parameter size and FLOPs.

5 Related Work

5.1 Network Pruning in Transformer

Pruning removes unimportant connections or neurons in the network. Compared with pruning connections (Yu et al., 2019; Gordon et al., 2020; Sanh



(a) After QK. (b) After V. (c) After O. (d) After FFN1. (e) After FFN2.

Figure 4: Candidate positions to add the ghost module in the Transformer layer.

Position	Convolution Type	FLOPs (G)	#params (M)	Avg. acc
`After QK`	Conv2D	5.4	45	75.9
`After V`	DWConv	3.6	39	74.3
`After O`	DWConv	2.0	32	74.4
`After FFN1`	DWConv	2.0	32	74.3
`After FFN2`	DWConv	2.0	32	74.3
Ours: `After O&FFN2`	DWConv	2.0	32	75.8

Table 6: Comparison of different positions to add the ghost module. Average development set accuracy on five GLUE tasks (RTE, SST-2, MRPC, CoLA and STS-B) are reported. The pruned BERT with width multiplier $1/12$ is used as backbone model .

et al., 2020), structured pruning prunes away a group of parameters without changing the model topology and is more favored for hardware and real inference speedup.

In the width direction, Michel et al. (2019); Voita et al. (2019) retain the performance after pruning a large percentage of attention heads in a structured manner. Besides attention heads, McCarley et al. (2019) also prune the neurons and the embeddings. In the depth direction, pruning Transformer layers is proposed in LayerDrop (Fan et al., 2019) via structured dropout. Efficient choice of Transformer layers at inference via early exit are also proposed in (Liu et al., 2020; Xin et al., 2020; Zhou et al., 2020). Hou et al. (2020) perform structured pruning in both width and depth directions. The importance of attention heads and neurons in the intermediate layer of Feed-forward network is measured by their impact on the loss, and the least important heads and neurons are pruned away.

5.2 Enhanced Representation in Transformer-based Models

Various methods have been proposed to use linear or convolution operations to enhance the representation of the Transformer layers.

The first group of research works replaces the self-attention mechanism or feed-forward networks with simpler and more efficient convolution operations, while maintaining comparable results. Wu et al. (2019) introduce the token-based dynamic depth-wise convolution to compute the importance of context elements, and achieve better results in various NLP tasks. Iandola et al. (2020) replace all the feed-forward networks with grouped convolution. AdaBERT (Chen et al., 2020) uses differentiable neural architecture to search for more efficient convolution-based NLP models.

The second group uses linear or convolutional module along with the self-attention mechanism for more powerful representation. The new module can be incorporated though *serial* connection to the original self-attention mechanism (Mehta et al., 2020), or be used in *parallel* with the original self-attention mechanism (Wu et al., 2020; Jiang et al., 2020) to capture both local and global context dependency. Serial and parallel connections of these linear or convolution operations to Transformer layers are also extended to multi-task (Houlsby et al., 2019; Stickland and Murray, 2019) and multi-lingual tasks (Pfeiffer et al., 2020).

Note that the proposed ghost modules are orthogonal to the above methods in that these modules are used to generate more features for the Transformer models and can be easily integrated into existing methods to boost their performance.

6 Conclusion

In this paper, we propose GhostBERT to generate more features in pre-trained model with cheap operations. We use the softmax-normalized 1-Dimensional Convolutions as ghost modules and add them to the output of the MHA and FFN of each Transformer layer. Empirical results on BERT, RoBERTa and ELECTRA demonstrate that adding the proposed ghost modules enhances the representation power and boosts the performance of the original model by supplying more features.

Acknowledgement

We thank MindSpore for the partial support of this work, which is a new deep learning computing framework. Given the superior performance of Huawei Ascend AI Processor and MindSpore computing framework, our code will be released based on MindSpore at (`https://gitee.com/mindspore/mindspore/tree/master/model_zoo/research/nlp/ghostbert`).

References

H. Bai, W. Zhang, L. Hou, L. Shang, J. Jin, X. Jiang, Q. Liu, M. Lyu, and I. King. 2021. Binarybert: Pushing the limit of bert quantization. In *Annual Meeting of the Association for Computational Linguistics*.

D. Chen, Y. Li, M. Qiu, Z. Wang, B. Li, B. Ding, H. Deng, J. Huang, W. Lin, and J. Zhou. 2020. Adabert: Task-adaptive BERT compression with differentiable neural architecture search. In *International Joint Conference on Artificial Intelligence*.

K. Clark, M. Luong, Q. Le, and C. Manning. 2020. ELECTRA: pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations*.

J. Devlin, M. Chang, K. Lee, and K. Toutanova. 2019. BERT: pre-training of deep bidirectional transformers for language understanding. In *North American Chapter of the Association for Computational Linguistics*.

A. Fan, E. Grave, and A. Joulin. 2019. Reducing transformer depth on demand with structured dropout. In *International Conference on Learning Representations*.

A. Fan, P. Stock, B. Graham, E. Grave, R. Gribonval, H. Jegou, and A. Joulin. 2020. Training with quantization noise for extreme model compression. Preprint arXiv:2004.07320.

M. A. Gordon, K. Duh, and N. Andrews. 2020. Compressing bert: Studying the effects of weight pruning on transfer learning. Preprint arXiv:2002.08307.

K. Han, Y. Wang, Q. Tian, J. Guo, C. Xu, and C. Xu. 2020. Ghostnet: More features from cheap operations. In *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1580–1589.

L. Hou, Z. Huang, L. Shang, X. Jiang, X. Chen, and Q. Liu. 2020. Dynabert: Dynamic bert with adaptive width and depth. In *Advances in Neural Information Processing Systems*.

N. Houlsby, A. Giurgiu, S. Jastrzebski, B. Morrone, Q. Laroussilhe, A. Gesmundo, M. Attariyan, and S. Gelly. 2019. Parameter-efficient transfer learning for NLP. In *International Conference on Machine Learning*.

F. Iandola, A. Shaw, R. Krishna, and K. Keutzer. 2020. Squeezebert: What can computer vision teach nlp about efficient neural networks? In *Proceedings of SustaiNLP: Workshop on Simple and Efficient Natural Language Processing*.

Z. Jiang, W. Yu, D. Zhou, Y. Chen, J. Feng, and S. Yan. 2020. Convbert: Improving BERT with span-based dynamic convolution. In *Advances in Neural Information Processing Systems*.

X. Jiao, Y. Yin, L. Shang, X. Jiang, X. Chen, L. Li, F. Wang, and Q. Liu. 2020. Tinybert: Distilling BERT for natural language understanding. In *Findings of Empirical Methods in Natural Language Processing*.

O. Kovaleva, A. Romanov, A. Rogers, and A. Rumshisky. 2019. Revealing the dark secrets of bert. In *Conference on Empirical Methods in Natural Language Processing*, pages 4356–4365.

Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut. 2020. Albert: A lite bert for self-supervised learning of language representations. In *International Conference on Learning Representations*.

W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. In *Annual Meeting of the Association for Computational Linguistics*.

Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. Preprint arXiv:1907.11692.

J. S. McCarley, R. Chakravarti, and A. Sil. 2019. Structured pruning of a bert-based question answering model. Preprint arXiv:1910.06360.

S. Mehta, M. Ghazvininejad, S. Iyer, L. Zettlemoyer, and H. Hajishirzi. 2020. Delight: Very deep and light-weight transformer. *CoRR*, abs/2008.00623.

P. Michel, O. Levy, and G. Neubig. 2019. Are sixteen heads really better than one? In *Advances in Neural Information Processing Systems*.

MindSpore. `https://www.mindspore.cn`. 2021.

J. Pfeiffer, I. Vulić, I. Gurevych, and S. Ruder. 2020. Mad-x: An adapter-based framework for multi-task cross-lingual transfer. In *Conference on Empirical Methods in Natural Language Processing*, pages 7654–7673.

A. Rogers, O. Kovaleva, and A. Rumshisky. 2020. A primer in bertology: What we know about how bert works. Preprint arXiv:2002.12327.

V. Sanh, L. Debut, J. Chaumond, and T. Wolf. 2019. Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. Preprint arXiv:1910.01108.

V. Sanh, T. Wolf, and A. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. In *Advances in Neural Information Processing Systems*, volume 33.

S. Shen, Z. Dong, J. Ye, L. Ma, Z. Yao, A. Gholami, M. W. Mahoney, and K. Keutzer. 2020. Q-bert: Hessian based ultra low precision quantization of bert. In *AAAI Conference on Artificial Intelligence*.

A. C. Stickland and I. Murray. 2019. Bert and pals: Projected attention layers for efficient adaptation in multi-task learning. In *International Conference on Machine Learning*, pages 5986–5995. PMLR.

S. Sun, Y. Cheng, Z. Gan, and J. Liu. 2019. Patient knowledge distillation for bert model compression. In *Conference on Empirical Methods in Natural Language Processing*.

Z. Sun, H. Yu, X. Song, R. Liu, Y. Yang, and D. Zhou. 2020. Mobilebert: a compact task-agnostic BERT for resource-limited devices. In *Annual Meeting of the Association for Computational Linguistics*.

E. Voita, D. Talbot, F. Moiseev, R. Sennrich, and I. Titov. 2019. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. In *Annual Meeting of the Association for Computational Linguistics*.

A. Wang, A. Singh, J. Michael, F. Hill, O. Levy, and S. Bowman. 2019. GLUE: A multi-task benchmark and analysis platform for natural language understanding. In *International Conference on Learning Representations*.

F. Wu, A. Fan, A. Baevski, Y. Dauphin, and M. Auli. 2019. Pay less attention with lightweight and dynamic convolutions. In *International Conference on Learning Representations*.

Z. Wu, Z. Liu, J. Lin, Y. Lin, and S. Han. 2020. Lite transformer with long-short range attention. In *International Conference on Learning Representations*.

J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *Annual Meeting of the Association for Computational Linguistics*.

H. Yu, S. Edunov, Y. Tian, and A. S Morcos. 2019. Playing the lottery with rewards and multiple languages: lottery tickets in rl and nlp. In *International Conference on Learning Representations*.

W. Zhang, L. Hou, Y. Yin, L. Shang, X. Chen, X. Jiang, and Q. Liu. 2020. Ternarybert: Distillation-aware ultra-low bit bert. In *Empirical Methods in Natural Language Processing*.

W. Zhou, C. Xu, T. Ge, J. McAuley, K. Xu, and F. Wei. 2020. Bert loses patience: Fast and robust inference with early exit. In *Advances in Neural Information Processing Systems*.

A Experiment Settings

A.1 Statistics of GLUE datasets

The GLUE benchmark (Wang et al., 2019) consists of various sentence understanding tasks, including two single-sentence classification tasks (CoLA and SST-2), three similarity and paraphrase tasks (MRPC, STS-B and QQP), and four inference tasks (MNLI, QNLI, RTE and WMLI). For MNLI task, we report the result on the matched section. For Winograd Schema (WNLI), it is a small natural inference dataset while even a majority baseline outperforms many methods on it. As is noted in the GLUE official website[1], there are some issues with the construction of it. Like previous work (Hou et al., 2020; Jiang et al., 2020), we do not experiment on WNLI. We use the default train/development/test splits from the official website.

Corpus	**Train**	**Test**	**Task**	**Metrics**
		Single-Sentence Tasks		
CoLA	8.5k	1k	acceptability	Matthews corr.
SST-2	67k	1.8k	sentiment	acc.
		Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc.
STS-B	7k	1.4k	sentence similarity	Spearman corr.
QQP	364k	391k	paraphrase	acc.
		Inference Tasks		
MNLI	393k	20k	NLI	matched acc.
QNLI	105k	5.4k	QA/NLI	acc.
RTE	2.5k	3k	NLI	acc.
WNLI	634	146	coreference/NLI	acc.

Table 7: Statistics of the GLUE datasets. All tasks are single-sentence or sentence-pair classification tasks, except STS-B, which is a regression task. MNLI has three classes while all other classification tasks have two.

A.2 Hyperparameters

We show the detailed hyperparameters for the distillation and fine-tuning stages in Section 2.3 of the proposed method on the GLUE benchmark in Table 8.

A.3 FLOPs

Floating-point operations (FLOPs) measures the number of floating-point operations that the model performs for a single process and can be used as a measure of the computational complexity of deep neural network models. To count the FLOPs, we follow the setting in (Hou et al., 2020) and infer FLOPs with batch size 1 and sequence length 128. Since the operations in the embedding lookup are relatively cheap compared to those in Transformer layers, following (Hou et al., 2020; Sun et al., 2020), we do not count them. Note that the reported FLOPs for ELECTRA (Clark et al., 2020) and ConvBERT(Jiang et al., 2020) in their original papers include those for the embedding lookup, and are slightly different from the numbers in this paper.

[1]https://gluebenchmark.com/faq

	Distillation	Fine-tuning
Batch Size	32	32
Learning Rate	$2e-5$	$2e-5$
Adam β_1	0.9	0.9
Adam β_2	0.999	0.999
Warmup Steps	0	0
Learning Rate Decay	Linear	Linear
Weight Decay	0	0
Gradient Clipping	1	1
Dropout	0.1	0.1
Attention Dropout	0.1	0.1
Distillation	Y	Y
$\lambda_1, \lambda_2, \lambda_3$	0, 1, 1	1, 0, 0
Training Epochs (MNLI, QQP)	1	3
Training Epochs (Other datasets)	3	3

Table 8: Hyperparameters for the distillation and fine-tuning stages in training GhostBERT on the GLUE benchmark.

B More Experiment Results

B.1 Full Results of Different Convolution Kernel Sizes

In Table 9, we show the detailed results of different convolutions kernel sizes for each of the five tasks (SST-2, MRPC, CoLA, STS-B and RTE). As can be seen, for each task, DWConv with kernel size 3 has the best performance.

m	Kernel Size	SST-2	MRPC	CoLA	STS-B	RTE	Avg.
3/12	1	92.5	85.3	**54.8**	89.1	70.8	78.5
	3	**94.6**	**86.5**	53.9	**89.2**	**72.9**	**79.4**
	5	92.7	**86.5**	53.5	89.0	70.4	78.4
	9	92.6	85.5	53.4	89.0	69.3	78.0
	17	92.4	84.8	53.3	88.9	68.2	77.5
1/12	1	92.1	85.3	41.7	87.6	64.3	74.2
	3	**92.8**	**86.0**	**46.1**	**87.8**	**66.1**	**75.8**
	5	92.2	85.3	41.1	87.5	64.6	74.2
	9	92.1	84.8	41.4	87.5	63.9	73.9
	17	92.0	84.3	40.9	87.5	63.5	73.7

Table 9: Comparison of different kernel sizes on the development set on five tasks of GLUE. m stands for the width multiplier.

m	Model	FLOPs(G)	#params(M)	MNLI	QNLI	QQP	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
$1/12$	Prune	2.0	32	82.0	88.9	90.4	60.7	91.2	83.1	42.6	86.4	78.2
	Ghost	2.0	32	82.8	90.0	90.5	66.1	92.8	86.0	46.1	87.8	**80.3**
$3/12$	Prune	5.8	46	83.6	91.1	90.9	68.6	92.6	85.1	53.7	88.5	81.8
	Ghost	5.8	46	84.3	91.6	91.4	72.9	94.6	86.5	53.9	89.2	**83.1**
$6/12$	Prune	11.3	67	82.9	90.6	90.9	71.8	92.1	86.8	54.2	88.4	82.2
	Ghost	11.3	67	84.7	92.2	91.2	72.2	92.9	87.3	58.1	89.2	**83.5**
$9/12$	Prune	16.9	88	84.2	91.5	91.0	72.2	92.4	86.8	58.4	89.4	83.2
	Ghost	16.9	88	84.8	92.1	91.2	72.6	92.6	87.5	61.1	89.8	**84.0**

Table 10: Development set results of the pruned BERT and GhostBERT. m is the width multiplier of the model.

Position	Convolution Type	FLOPs(G)	#params(M)	RTE	SST-2	MRPC	CoLA	STS-B	Avg.
`After QK`	Conv2D	5.4	45	65.3	91.9	85.8	48.5	87.9	75.9
`After V`	DWConv	3.6	39	62.1	91.6	86 7	44.3	87.7	74.3
`After O`	DWConv	2.0	32	63.2	91.7	85.5	43.8	87.8	74.4
`After FFN1`	DWConv	2.0	32	64.6	91.2	84.6	43.3	87.7	74.3
`After FFN2`	DWConv	2.0	32	62.5	91.5	84.8	44.9	87.7	74.3
Ours: `After O&FFN2`	DWConv	2.0	32	66.1	92.8	86.0	46.1	87.8	75.8

Table 11: Comparison of different ghost positions on the development set on five tasks of GLUE. BERT-base is set as backbone model with the width multiplier 1/12.

B.2 Full Results of Pruned BERT

In Table 10, we show the detailed results of the pruned BERT and the GhostBERT for each task. We can see that under the same training procedure, the GhostBERT outperforms the pruned BERT over all compared sizes.

B.3 Full Results of Different Positions

Table 11 shows the detailed results of adding ghost modules to different positions of the model.

B.4 Generating More Features

As is mentioned at the end of Section 2.3, we generate only one ghost feature for each MHA and FFN, i.e., $F = 1$ to save computation and memory. Indeed, our framework has no limitation on F, and also allows the model to generate more features (i.e., $F > 1$). In this section, we discuss the relationship between generating more ghost features and the computation/memory requirements.

Following the notation in Section 2 and omitting the cheap computation of ReLU and softmax, generating F ghost features from M features for all L layers requires $2LMFdk$ additional parameters and $4LMFndk$ additional FLOPs. Both of them scale linearly as F, and can be large when F is large. For instance, for BERT-base with $d = 768$, when $n = 128$, $k = 3$, $M = 12$ and $F = 12$, the additional #parameters and FLOPs are $8M$ and $2.0G$ respectively, accounting for 7.2% and 9.1% of the backbone model.

When F increases, the accuracy of GhostBERT first increases slowly and soon begins to saturate or decrease. E.g., for GhostBERT ($m = 1/12$), the average development accuracy on GLUE only increases from 80.3 to 80.6 when F increases from 1 to 4, and then saturates when $F > 4$. For GhostBERT ($m = 3/12$), the highest accuracy 83.1 is achieved when $F = 1$ or 2, and then the accuracy begins to decrease.

Thus in the paper, we simply choose F=1 which is cheap, but already achieves good performance on most tasks.