SqueezeBERT: What can computer vision teach NLP about efficient neural networks?

Forrest N. Iandola
forresti@berkeley.edu

Albert E. Shaw
ashaw596@gmail.com

Ravi Krishna
UC Berkeley EECS
ravi.krishna@berkeley.edu

Kurt W. Keutzer
UC Berkeley EECS
keutzer@berkeley.edu

Abstract

Humans read and write hundreds of billions of messages every day. Further, due to the availability of large datasets, large computing systems, and better neural network models, natural language processing (NLP) technology has made significant strides in understanding, proofreading, and organizing these messages. Thus, there is a significant opportunity to deploy NLP in myriad applications to help web users, social networks, and businesses. Toward this end, we consider smartphones and other mobile devices as crucial platforms for deploying NLP models at scale. However, today's highly-accurate NLP neural network models such as BERT and RoBERTa are extremely computationally expensive, with BERT-base taking 1.7 seconds to classify a text snippet on a Pixel 3 smartphone. To begin to address this problem, we draw inspiration from the computer vision community, where work such as MobileNet has demonstrated that grouped convolutions (e.g., depthwise convolutions) can enable speedups without sacrificing accuracy. We demonstrate how to replace several operations in self-attention layers with grouped convolutions and use this technique in a novel network architecture called SqueezeBERT, which runs 4.3x faster than BERT-base on the Pixel 3 while achieving competitive accuracy on the GLUE test set.

A PyTorch-based implementation of SqueezeBERT is available as part of the Hugging Face Transformers library: https://huggingface.co/squeezebert

1 Introduction and Motivation

The human race writes over 300 billion messages per day (Sayce, 2019; Schultz, 2019; Al-Heeti, 2018; Templatify, 2017). Out of these, more than half of the world's emails are read on mobile devices, and nearly half of Facebook users exclusively access Facebook from a mobile device (Lovely Mobile News, 2017; Donnelly, 2018). Natural language processing (NLP) technology has the potential to aid these users and communities in several ways. When a person writes a message, NLP models can help with spelling and grammar checking as well as sentence completion. When content is added to a social network, NLP can facilitate content moderation before it appears in other users' news feeds. When a person consumes messages, NLP models can help classify messages into folders, compose news feeds, prioritize messages, and identify duplicates.

In recent years, the development and adoption of Attention Neural Networks have led to dramatic improvements in almost every area of NLP. In 2017, Vaswani *et al.* proposed the multi-head self-attention module, which demonstrated superior accuracy to recurrent neural networks on English-German machine language translation (Vaswani et al., 2017).[1] These modules have since been adopted by GPT (Radford et al., 2018) and BERT (Devlin et al., 2019) for sentence classification, and by GPT-2 (Radford et al., 2019) and CTRL (Keskar et al., 2019) for sentence completion and generation. Recent works such as ELECTRA (Clark et al., 2020) and RoBERTa (Liu et al., 2019) have shown that larger datasets and more sophisticated training regimes can further improve the accuracy of self-attention networks.

Considering the enormity of the textual data created by humans on mobile devices, a natural approach is to deploy the NLP models directly onto mobile devices, embedding them in the apps used to read, write, and share text. Unfortunately, highly-accurate NLP models are computationally expensive, making mobile deployment impractical. For example, we observe that running the BERT-base

[1]Neural networks that use the self-attention modules of Vaswani *et al.* are sometimes called "Transformers," but in the interest of clarity, we call them "self-attention networks."

network on a Google Pixel 3 smartphone approximately 1.7 seconds to classify a single text data sample.[2] Much of the research on efficient self-attention networks for NLP has just emerged in the past year. However, starting with SqueezeNet (Iandola et al., 2016b), the mobile computer vision (CV) community has spent the last four years optimizing neural networks for mobile devices. Intuitively, it seems like there must be opportunities to apply the lessons learned from the rich literature of mobile CV research to accelerate mobile NLP. In the following, we review what has already been applied and propose two additional techniques from CV that we will leverage to accelerate NLP models.

1.1 What has CV research already taught NLP research about efficient networks?

In recent months, novel self-attention networks have been developed with the goal of achieving faster inference. At present, the MobileBERT network defines the state-of-the-art in low-latency text classification for mobile devices (Sun et al., 2020). MobileBERT takes approximately 0.6 seconds to classify a text sequence on a Google Pixel 3 smartphone while achieving higher accuracy on the GLUE benchmark, which consists of 9 natural language understanding (NLU) datasets (Wang et al., 2018), than other efficient networks such as DistilBERT (Sanh et al., 2019), PKD (Sun et al., 2019a), and several others (Lan et al., 2019; Turc et al., 2019; Jiao et al., 2019; Xu et al., 2020). To achieve this, MobileBERT introduced two concepts into their NLP self-attention network that are already in widespread use in CV neural networks:

1. **Bottleneck layers.** In ResNet (He et al., 2016), the 3x3 convolutions are computationally expensive, so a 1x1 "bottleneck" convolution is employed to reduce the number of channels input to each 3x3 convolution layer. Similarly, MobileBERT adopts bottleneck layers that reduce the number of channels before each self-attention layer, reducing the computational cost of the self-attention layers.

2. **High-information flow residual connections.** In BERT-base, the residual connections serve as links between the low-channel-count (768 channels) layers. The high-channel-count (3072 channels) layers in BERT-base do not have residual connections. However, the ResNet and Residual-SqueezeNet (Iandola et al., 2016b) CV networks connect the high-channel-count layers with residuals, enabling higher information flow through the network. Similar to these CV networks, MobileBERT adds residual connections between the high-channel-count layers.

1.2 What else can CV research teach NLP research about efficient networks?

We are encouraged by the progress that MobileBERT has made in leveraging ideas that are popular in the CV literature to accelerate NLP. However, we are aware of two other ideas from CV, which weren't used in MobileBERT which could be applied to accelerate NLP:

1. **Convolutions.** Since the 1980s, computer vision neural nets have relied heavily on convolutional layers (Fukushima, 1980; LeCun et al., 1989). Convolutions are quite flexible and well-optimized in software, and they can implement things as simple as a 1D fully-connected layer, or as complex as a 3D dilated layer that performs upsampling or downsampling.

2. **Grouped convolutions.** A popular technique in modern mobile-optimized neural networks is grouped convolutions (see Section 3). Proposed by Krizhevsky *et al.* in the 2012 winning submission to the ImageNet image classification challenge (Krizhevsky et al., 2011, 2012; Russakovsky et al., 2015), grouped convolutions disappeared from the literature from some years, then re-emerged as a key technique circa 2016 (Chollet, 2016; Xie et al., 2017) and today are extensively used in efficient CV networks such as MobileNet (Howard et al., 2017), ShuffleNet (Zhang et al., 2018), and EfficientNet (Tan and Le, 2019). While common in CV literature, we are not aware of work applying grouped convolutions to NLP.

1.3 SqueezeBERT: Applying lessons learned from CV to NLP

In this work, we describe how to apply convolutions and particularly grouped convolutions in the design of a novel self-attention network for NLP,

[2]Note that BERT-base (Devlin et al., 2019), RoBERTa-base (Liu et al., 2019), and ELECTRA-base (Clark et al., 2020) all use the same self-attention encoder architecture, and therefore these networks incur approximately the same latency on a smartphone.

which we call SqueezeBERT. Empirically, we find that SqueezeBERT runs at lower latency on a smartphone than BERT-base, MobileBERT, and several other efficient NLP models, while maintaining competitive accuracy.

2 Implementing self-attention with convolutions

In this section, first, we review the basic structure of self-attention networks. Next, we identify that their biggest computational bottleneck is in their position-wise fully-connected (PFC) layers. We then show that these PFC layers are equivalent to a 1D convolution with a kernel size of 1.

2.1 Self-attention networks

In most BERT-derived networks there are typically 3 stages: the embedding, the encoder, and the classifier (Devlin et al., 2019; Liu et al., 2019; Clark et al., 2020; Sun et al., 2020; Lan et al., 2019).[3] The embedding converts preprocessed words (represented as integer-valued tokens) into learned feature-vectors of floating-point numbers. The encoder is comprised of a series of self-attention and other layers. The classifier produces the network's final output. As we will see later in Table 1, the embedding and the classifier account for less than 1% of the runtime of a self-attention network, so we focus our discussion on the encoder.

We now describe the encoder that is used in BERT-base (Devlin et al., 2019). The encoder consists of a stack of blocks. Each block consists of a three position-wise fully-connected (PFC) layers, then a self-attention module, and finally a stack of three position-wise fully-connected layers, known as feed-forward network (FFN) layers. The initial three PFC layers, are used to generate the *query* (Q), *key* (K), and *value* (V) activation vectors for each position in the feature embedding. Each of these Q, K, and V layers applies the same operation to each position in the feature embedding independently. While neural networks traditionally multiply weights by activations, a distinguishing factor of attention neural networks is that they multiply activations by other activations, enabling dynamic weighting of tensor elements to adjust based on the input data. Further, attention networks allow modeling of arbitrary dependencies regardless of their distance in the input or output (Vaswani et al., 2017). The self-attention module proposed by Vaswani *et al.* (Vaswani et al., 2017) (which is also used by GPT (Radford et al., 2018), BERT (Devlin et al., 2019), RoBERTa (Liu et al., 2019), ELECTRA (Clark et al., 2020) and others) multiplies the Q, K, and V activations together using the equation $softmax(\frac{QK^T}{\sqrt{d_k}})V$, where d_k is the number of channels in one attention head.[4]

[3]Some self-attention networks such as (Vaswani et al., 2017; Radford et al., 2018) also have "decoder” stage. The decoder typically uses a similar neural architecture as the encoder, but is auto-regressive.

Table 1: **How does BERT spend its time?** This is a breakdown of computation (in floating-point operations, or FLOPs) and latency (on a Google Pixel 3 smartphone) in BERT-base. The sequence length is 128.

Stage	Module type	FLOPs	Latency
Embedding	Embedding	0.00%	0.26%
Encoder	Self-attention calculations	2.70%	11.3%
Encoder	PFC layers	97.3%	88.3%
Final Classifier	PFC layers	0.00%	0.02%
Total		100%	100%

2.2 Benchmarking BERT for mobile inference

To identify the parts of BERT that are time-consuming to compute, we profile BERT on a smartphone. Specifically, we measure the neural network's latency using PyTorch (Paszke et al., 2019) and TorchScript on a Google Pixel 3 smartphone, with an input sequence length of 128 and a batch size of 1. This is a reasonable sequence length for text messages, instant messages, short emails, and other messages that are commonly written and read by smartphone users. In Table 1, we show the breakdown of FLOPs and latency among the main components of the BERT network, and we observe that the self-attention calculations (i.e. $softmax(\frac{QK^T}{\sqrt{d_k}})V$) account for only 11.3% of the total latency. However, PFC layers account for 88.3% of the latency.

2.3 Replacing the position-wise fully connected (PFC) layers with convolutions

Given that PFC layers account for the overwhelming majority of the latency, we now focus on reducing the PFC layers' latency. In particular, we intend to replace the PFC layers with grouped convolutions, which have been shown to produce significant speedups in computer vision networks. As

[4]For example, in BERT-base, the self-attention module has 768 channels and 12 heads, so $d_k = \frac{768}{12} = 64$.

a first step in this direction, we now show that the position-wise fully-connected layers used throughout the BERT encoder are a special case of non-grouped 1D convolution.

Let $\mathbf{w}$ denote the weights of the position-wise fully-connected layer with dimensions ($\mathcal{C}$, $\mathcal{C}$). Given an input feature vector $\mathbf{f}$ of dimensions $(\mathcal{P}, \mathcal{C})$ with P positions and $\mathcal{C}$ channels to generate an output of $(P, \mathcal{C})$ features, the operation performed by the position-wise fully-connected layer for each output channel c at position p can be defined:

$$PFC_{p,c}(\mathbf{f}, \mathbf{w}) = \sum_{i}^{\mathcal{C}} \mathbf{w}_{c,i} * \mathbf{f}_{p,i}$$

Then if we consider the definition of a 1D convolution with kernel size $\mathcal{K}$ with the same input and output dimensions. Let $\mathbf{q}$ be the weights of the convolution with with dimensions ($\mathcal{C}$, $\mathcal{C}$, $\mathcal{K}$)

$$Conv_{p,c}(\mathbf{f}, \mathbf{q}) = \sum_{i}^{\mathcal{C}} \sum_{k}^{K} \mathbf{q}_{c,i,k} * \mathbf{f}_{(p - \frac{\mathcal{K}-1}{2} + k),i}$$

we observe that the position-wise fully-connected operation is equivalent to a convolution with a kernel size of $\mathcal{K} = 1$ where $\mathbf{q}_{c,i,0} = \mathbf{w}_{c,i}$

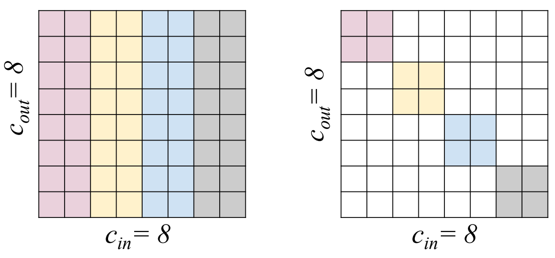
$$Conv_{p,c}(\mathbf{f}, \mathbf{q}) = \sum_{i}^{\mathcal{C}} \mathbf{q}_{c,i,0} * \mathbf{f}_{p,i}$$

Thus, the PFC layers of Vaswani *et al.* (Vaswani et al., 2017), GPT, BERT, and similar self-attention networks can be implemented using convolutions without changing the networks' numerical properties or behavior.

3 Incorporating grouped convolutions into self-attention

Now that we have shown how to implement the expensive PFC layers in self-attention networks using convolutions, we can incorporate efficient grouped convolutions into a self-attention network. Grouped convolutions are defined as follows.

Given an input feature vector of dimensions $(\mathcal{P}, \mathcal{C})$ with $\mathcal{P}$ positions and $\mathcal{C}$ channels outputting a vector with dimensions $(\mathcal{P}, \mathcal{C})$, a 1d convolution with kernel size $\mathcal{K} = 1$ and } groups and weight vector $\mathbf{q}$ of dimensions $(\mathcal{C}, \frac{\mathcal{C}}{g})$ can be defined as follows. Let $\mathcal{N} = \frac{\mathcal{C}}{g}$ where $\mathcal{N}$ is the number of channels in each group.



(a) groups=1 (b) groups=4

Figure 1: **Traditional vs. grouped convolutions.** In panel (a), we illustrate the weight matrix of a traditional 1D convolution with 8 input channels, 8 output channels, and a kernel size of 1. In panel (b), we illustrate a grouped convolution with $g = 4$. White cells in the grid are empty. Observe that with $g = 4$, the weight matrix has one-fourth the number of parameters as a traditional convolution.

$$GConv_{p,c}(\mathbf{f}, \mathbf{q}) = \sum_{i}^{\frac{\mathcal{C}}{g}} \mathbf{q}_{c,i,0} * \mathbf{f}_{p,(i + \lfloor \frac{c}{\mathcal{N}} \rfloor \mathcal{N})}$$

This is equivalent to splitting the the input vector into g separate vectors of size $(P, \frac{C_{in}}{g})$ along the C dimension and running g separate convolutions with independent weights each computing vectors of size $(P, \frac{C_{out}}{g})$. The grouped convolution, however, requires only $\frac{1}{g}$ as many floating-point operations (FLOPs) and $\frac{1}{g}$ as many weights as an ordinary convolution, not counting the small (and unchanged) amount of operations needed for the channel-wise bias term that is often included in convolutional layers.[5] Finally, to complement the mathematical explanation of grouped convolutions, we illustrate the difference between traditional convolutions and grouped convolutions in Figure 1.

3.1 SqueezeBERT

Now, we describe our proposed neural architecture called SqueezeBERT, which uses grouped convolutions. SqueezeBERT is much like BERT-base, but with PFC layers implemented as convolutions, and grouped convolutions for many of the layers. Recall from Section 2 that each block in the BERT-base encoder has a self-attention module that ingests the activations from 3 PFC layers, and the block also has 3 more PFC layers called feed-forward network layers (FFN$_1$, FFN$_2$, and FFN$_3$). The FFN layers have the following dimensions: FFN$_1$ has $C_{in} = C_{out} = 768$, FFN$_2$

[5]Note that the grouped convolution with $g = 1$ is identical to an ordinary convolution.

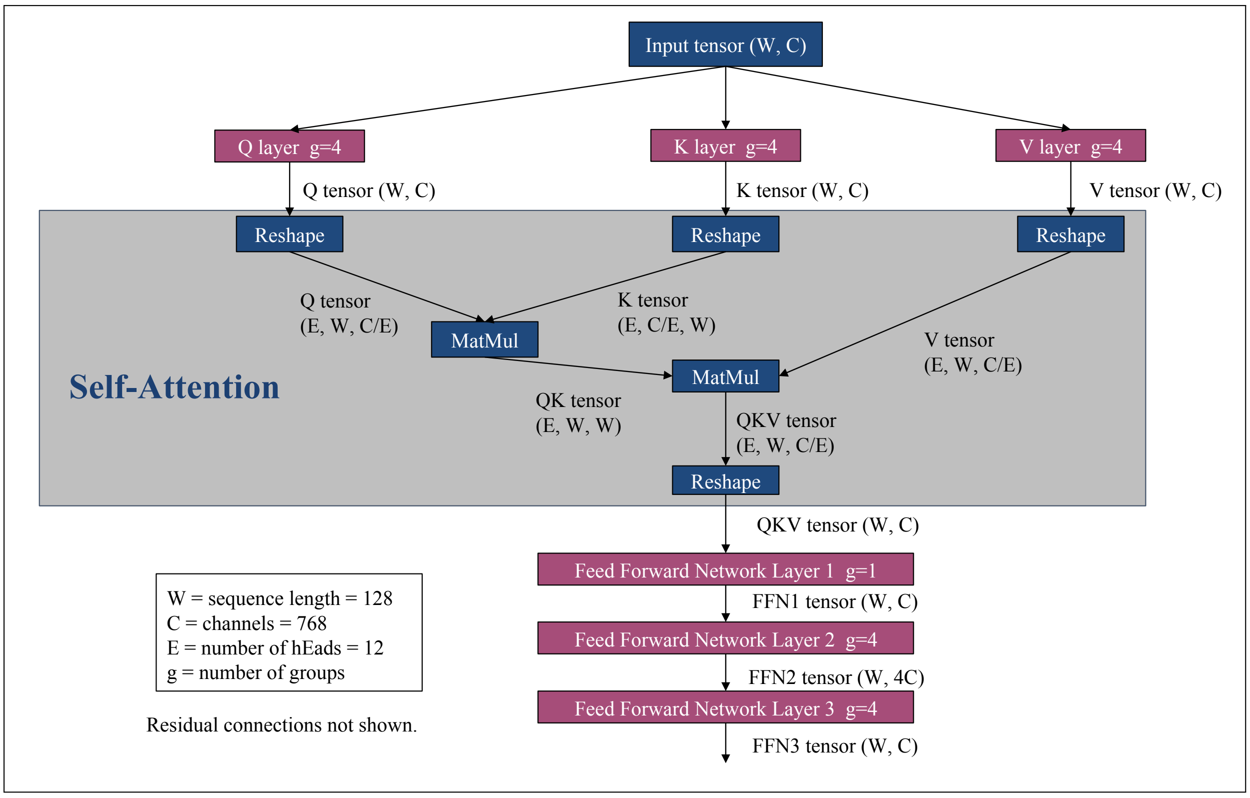


Figure 2: **One block of the SqueezeBERT encoder.** The SqueezeBERT encoder consists of a stack of 12 of these modules.

has $C_{in} = 768$ and $C_{out} = 3072$, and FFN_3 has $C_{in} = 3072$ and $C_{out} = 768$. In all PFC layers of the self-attention modules, and in the FFN_2 and FFN_3 layers, we use grouped convolutions with $g = 4$. To allow for mixing across channels of different groups, we use $g = 1$ in the less-expensive FFN_1 layers. Note that in BERT-base, FFN_2 and FFN_3 each have 4 times more arithmetic operations than FFN_1. However, when we use $g = 4$ in FFN_2 and FFN_3, now all FFN layers have the same number of arithmetic operations. We illustrate one block of the SqueezeBERT encoder in Figure 2.

Finally, the embedding size (768), the number of blocks in the encoder (12), the number of heads per self-attention module (12), the WordPiece tokenizer (Schuster and Nakajima, 2012; Wu et al., 2016), and other aspects of SqueezeBERT are adopted from BERT-base. Aside from the convolution-based implementation and the adoption of grouped convolutions, the SqueezeBERT architecture is identical to BERT-base.

4 Experimental Methodology

4.1 Datasets

Pretraining Data. For pretraining, we use a combination of Wikipedia and BooksCorpus (Zhu et al., 2015), setting aside 3% of the combined dataset as a test set. Following the ALBERT paper, we use Masked Language Modeling (MLM) and Sentence Order Prediction (SOP) as pretraining tasks (Lan et al., 2019).

Finetuning Data. We finetune and evaluate SqueezeBERT (and other baselines) on the General Language Understanding Evaluation (GLUE) set of tasks. This benchmark consists of a diverse set of 9 NLU tasks; thanks to the structure and breadth of these tasks (see supplementary material for detailed task-level information), GLUE has become the standard evaluation benchmark for NLP research. A model's performance across the GLUE tasks likely provides a good approximation of that model's generalizability (especially for text classification tasks).

4.2 Training Methodology

Many recent papers on efficient NLP networks report results on models trained with bells and whistles such as distillation, adversarial training, and/or transfer learning across GLUE tasks. However, there is no standardization of these training schemes across different papers, making it difficult to distinguish the contribution of the model from the contribution of the training scheme to the final accuracy number. Therefore, we first train SqueezeBERT using a simple training scheme (described in Section 4.2.1, with results reported in Section 5.1), and then we train SqueezeBERT with distillation and other techniques (described in Section 4.2.2, with results reported in Section 5.2).

4.2.1 Training without bells and whistles

We pretrain SqueezeBERT from scratch (without distillation) using the LAMB optimizer, and we employ the hyperparameters recommended by the LAMB authors: a global batch size of 8192, a learning rate of 2.5e-3, and a warmup proportion of 0.28 (You et al., 2020). Following the LAMB paper's recommendations, we pretrain for 56k steps with a maximum sequence length of 128 and then for 6k steps with a maximum sequence length of 512.

For finetuning, we use the AdamW optimizer with a batch size of 16 without momentum or weight decay with $\beta_1 = 0.9$ and $\beta_2 = 0.999$ (Loshchilov and Hutter, 2019). As is common in the literature, during finetuning for each task, we perform hyperparameter tuning on the learning rate and dropout rate. We present more details on this in the supplementary material. In the interest of a fair comparison, we also train BERT-base using the aforementioned pretraining and finetuning protocol.

4.2.2 Training with bells and whistles

We now review recent techniques for improving the training of NLP networks, and we describe the approaches that we will use for the training and evaluation of SqueezeBERT in Section 5.2.

Distillation approaches used in other efficient NLP networks. While the term "knowledge distillation" was coined by Hinton *et al.* to describe a specific method and equation (Hinton et al., 2015), the term "distillation" is now used in reference to a diverse range of approaches where a "student" network is trained to replicate a "teacher" network. Some researchers distill only the final layer of the network (Sanh et al., 2019), while others also distill the hidden layers (Sun et al., 2019a, 2020; Xu et al., 2020). When distilling the hidden layers, some apply layer-by-layer distillation warmup, where each module of the student network is distilled independently while downstream modules are frozen (Sun et al., 2020). Some distill during pretraining (Sun et al., 2020; Sanh et al., 2019), some distill during finetuning (Xu et al., 2020), and some do both (Sun et al., 2019a; Jiao et al., 2019).

Bells and whistles used for training SqueezeBERT (for results in Section 5.2). Distillation is not a central focus of this paper, and there is a large design space of potential approaches to distillation, so we select a relatively simple form of distillation for use in SqueezeBERT training. We apply distillation only to the final layer, and only during finetuning. On the GLUE sentence classification tasks, we use soft cross entropy loss with respect to a weighted sum of the teacher's logits (Ψ_t) and a one-hot encoding of the ground-truth (Ψ_g). The weighting between the teacher logits and the ground-truth is controlled by a hyperparameter α. Formally, we write this weighted sum as:

$$\Psi = (1 - \alpha)\Psi_t + \alpha\Psi_g$$

Also note that GLUE has one regression task (STS-B text similarity), and for this task we replace the soft cross entropy loss with mean squared error. In addition to distillation, inspired by STILTS (Phang et al., 2018) and ELECTRA (Clark et al., 2020), we apply transfer learning from the MNLI GLUE task to other GLUE tasks as follows. The SqueezeBERT student model is pretrained using the approach described in Section 4.2.1, and then it is finetuned on the MNLI task. The weights from MNLI training are used as the initial student weights for other GLUE tasks except for CoLA.[6] Similarly, the teacher model is a BERT-base model that is pretrained using the ELECTRA method and then finetuned on MNLI. The teacher model is then finetuned independently on each GLUE task, and these task-specific teacher weights are used for distillation.

5 Results

We now turn our attention to comparing SqueezeBERT to other efficient neural networks.

5.1 Results *without* bells and whistles

In the upper portions of Tables 2 and 3, we compare our results to other efficient networks on the dev and test sets of the GLUE benchmark. Note that relatively few of the efficiency-optimized networks report results without bells and whistles, and most such results are reported on the development

[6]For CoLA, the student weights are pretrained (per Section 4.2.1) but not finetuned on MNLI prior to task-specific training.

Table 2: Comparison of neural networks on the **development** set of the GLUE benchmark. For tasks that have 2 metrics (e.g. MRPC's metrics are Accuracy and F1), we report the average of the 2 metrics. †denotes models trained by the authors of the present paper. Bells and whistles are: *A* = adversarial training; *D* = distillation of final layer; *E* = distillation of encoder layers; *S* = transfer learning across GLUE tasks (a.k.a. STILTs (Phang et al., 2018)); *W* = per-layer warmup. In GLUE accuracy, a dash means that accuracy for this task is not provided in the literature.

		GLUE accuracy										efficiency			
Model	Bells & Whistles	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	Average	#MParams	GFLOPs	Latency (ms)	Speedup
		*Results **without** bells and whistles*													
BERT-base†	-	85.2	84.8	89.9	92.2	92.7	62.8	90.7	91.2	76.5	85.1	109	22.5	1690	1.0x
MobileBERT (Sun et al., 2020)	-	80.8	-	-	88.2	90.1	-	-	84.3	-	-	25.3	5.36	572	3.0x
ALBERT-base (Lan et al., 2019)	-	81.6	-	-	-	90.3	-	-	-	-	-	12.0	22.5	1690	1.0x
SqueezeBERT†	-	82.3	82.9	89.4	90.5	92.0	53.7	89.4	89.8	71.8	82.4	51.1	7.42	390	4.3x
		*Results **with** bells and whistles*													
DistilBERT 6/768 (Sanh et al., 2019)	D	82.2	-	88.5	89.2	91.3	51.3	86.9	87.5	59.9	-	66	11.3	814	2.1x
Turc 6/768 (Turc et al., 2019)	D	82.5	83.4	89.6	89.4	91.1	-	-	87.2	66.7	-	67.5	11.3	814	2.1x
Theseus 6/768 (Xu et al., 2020)	DESW	82.3	-	89.6	89.5	91.5	51.1	88.7	89.0	68.2	-	66	11.3	814	2.1x
MobileBERT (Sun et al., 2020)	DEW	84.4	-	-	91.5	92.5	-	-	87.0	-	-	25.3	5.36	572	3.0x
SqueezeBERT†	DS	82.5	82.9	89.5	90.9	92.2	53.7	90.3	92.0	80.9	84.0	51.1	7.42	390	4.3x

Table 3: Comparison of neural networks on the **test** set of the GLUE benchmark. †denotes models trained by the authors of the present paper. Bells and whistles are: *A* = adversarial training; *D* = distillation of final layer; *E* = distillation of encoder layers; *S* = transfer learning across GLUE tasks (a.k.a. STILTs (Phang et al., 2018)); *W* = per-layer warmup.

		GLUE accuracy											efficiency			
Model	Bells & Whistles	MNLI-m	MNLI-mm	QQP	QNLI	SST-2	CoLA	STS-B	MRPC	RTE	WNLI	GLUE score	#MParams	GFLOPs	Latency (ms)	Speedup
		*Results **without** bells and whistles*														
BERT-base†	-	84.4	84.2	80.5	91.4	92.8	51.3	86.9	87.9	70.7	65.1	79.0	109	22.5	1690	1.0x
BERT-base (Devlin et al., 2019)	-	84.6	83.4	80.2	90.5	93.5	52.1	86.5	86.9	66.4	65.1	78.3	109	22.5	1690	1.0x
SqueezeBERT†	-	82.0	81.1	80.1	90.1	91.0	46.5	84.9	86.1	66.7	65.1	76.9	51.1	7.42	390	4.3x
		*Results **with** bells and whistles*														
TinyBERT 4/312 (Jiao et al., 2019)	DE	82.5	81.8	-	87.7	92.6	43.3	79.9	-	62.9	65.1	-	14.5	1.2	118	14x
ELECTRA-Small++ (Clark et al., 2020)	AS	81.6	-	-	88.3	91.1	55.6	84.6	84.9	63.6	65.1	-	14.0	2.62	248	6.8x
PKD 6/768 (Sun et al., 2019a)	DE	81.5	81.0	79.8	89.0	92.0	-	-	82.5	-	65.1	-	67.0	11.3	814	2.1x
Turc 6/768 (Turc et al., 2019)	D	82.8	82.2	79.7	89.4	91.8	-	-	84.3	65.3	65.1	-	67.5	11.3	814	2.1x
Theseus 6/768 (Xu et al., 2020)	DESW	82.4	82.1	80.5	89.6	92.2	47.8	84.9	85.4	66.2	65.1	77.1	66	11.3	814	2.1x
MobileBERT (Sun et al., 2020)	DEW	84.3	83.4	79.4	91.6	92.6	51.1	85.5	86.7	70.4	65.1	78.5	25.3	5.36	572	3.0x
SqueezeBERT†	DS	82.0	81.1	80.3	90.1	91.4	46.5	86.7	87.8	73.2	65.1	78.1	51.1	7.42	390	4.3x

(not test) set of GLUE. Fortunately, the authors of MobileBERT – a network which we will find in the next section compares favorably to other efficient networks with bells and whistles enabled – do provide development-set results without distillation on 4 of the GLUE tasks.[7] We observe in the upper portion of Table 2 that, when both networks are trained without distillation, SqueezeBERT achieves higher accuracy than MobileBERT on all of these tasks. This provides initial evidence that the techniques from computer vision that we have adopted can be applied to NLP, and reasonable accuracy can be obtained. Further, we observe that SqueezeBERT is 4.3x faster than BERT-base, while MobileBERT is 3.0x faster than BERT-base.[8]

Due to the dearth of efficient neural network results on GLUE without bells and whistles, we also provide a comparison in Table 2 with the ALBERT-base network. ALBERT-base is a version of BERT-base that uses the same weights across multiple attention layers, and it has a smaller encoder than BERT. Due to these design choices, ALBERT-base has 9x fewer parameters than BERT-base. However, ALBERT-base and BERT-base have the same number of FLOPs, and we observe in our measurements in Table 2 that ALBERT-base does not offer a speedup over BERT-base on a smartphone.[9] Further, on the two GLUE tasks where the ALBERT authors reported the accuracy of ALBERT-base, MobileBERT and SqueezeBERT both outperform the accuracy of ALBERT-base.

5.2 Results *with* bells and whistles

Now, we turn our attention to comparing SqueezeBERT to other models, all trained with bells-and-whistles. Note that the bells-and-whistles come at the cost of extra training time, but the bells-and-whistles do not change the inference time or model-size. In the lower portion of Table 3, we first observe that when trained with bells-and-whistles MobileBERT matches or outperforms the accuracy of the other efficient models (except SqueezeBERT) on 8 of the 9 GLUE tasks. Further, on 4 of the 9 tasks SqueezeBERT outperforms the accuracy of MobileBERT; on 4 of 9 tasks MobileBERT outperforms SqueezeBERT; and on 1 task (WNLI) all models predict the most frequently occurring category.[10] Also, SqueezeBERT achieves an average score across all GLUE tasks that is within 0.4 percentage-points of MobileBERT. Given the speedup of SqueezeBERT over MobileBERT, we think it is reasonable to say that SqueezeBERT and MobileBERT each offer a compelling speed-accuracy tradeoff for NLP inference on mobile devices.

6 Related Work

Quantization and Pruning. Quantization is a family of techniques which aims to reduce the number of bits required to store each parameter and/or activation in a neural network, while at the same time maintaining the accuracy of that network. This has been successfully applied to NLP in such works as (Shen et al., 2020; Zafrir et al., 2019). Pruning aims to directly eliminate certain parameters from the network while maintaining accuracy, thereby reducing the storage and potentially computational cost of that network; for an application of this to NLP, please see Sanh et al. (2020). These methods could be applied to SqueezeBERT to yield further efficiency improvements, but quantization and pruning are not a focus of this paper.

Addressing long sequence-lengths. In work such as SqueezeBERT and MobileBERT, the inference FLOPs and latency are evaluated using a sequence length of 128. This is a reasonable sequence length for use-cases such as classifying text messages, instant-messages, and short emails. However, if the goal is to classify longer-form texts such as book chapters or even an entire book, then the typical sequence length is much longer. While the positionwise fully-connected (PFC) layers in BERT scale linearly in the sequence length, the self-attention calculations scale quadratically in the sequence length. So, when classifying a long sequence, the self-attention calculations are the dominant factor in the FLOPs and latency of the neural

[7] Note that some papers report results on only the development set or the test set, and some papers only report results on a subset of GLUE tasks. Our aim with this evaluation is to be as inclusive as possible, so we include papers with incomplete GLUE results in our results tables.

[8] In our measurements, we find MobileBERT takes 572ms to classify one length-128 sequence on a Pixel 3 phone. This is slightly faster than the 620ms reported by the MobileBERT authors in the same setting (Sun et al., 2019b). We use the faster number in our comparisons. Further, all latencies in our results tables were benchmarked by us.

[9] However, reducing the number of parameters while retaining a high number of FLOPs can present other advantages, such as faster distributed training (Lan et al., 2019; Iandola et al., 2016a) and superior energy-efficiency (Iandola and Keutzer, 2017).

[10] Note that data augmentation approaches have been proposed to improve accuracy on WNLI; see (Kocijan et al., 2019). For fairness in comparing against our baselines, we choose not to use data augmentation to improve WNLI results.

network. Several recent projects have worked to address this problem. For instance, Funnel Transformer downsamples the sequence length in the first few layers of the network, and it upsamples the sequence length in the final few layers of the network (Dai et al., 2020). This approach is similar to computer vision models for semantic segmentation such as U-Net (Ronneberger et al., 2015). In addition, Longformer reduces the number of FLOPs by introducing structured sparsity into the self-attention tensors (Beltagy et al., 2020). Further, Linformer projects long sequences into shorter fixed-length sequences (Wang et al., 2020b). Finally, Tay et al. (2020) provide an extensive survey of approaches for redesigning self-attention networks to efficiently classify long sequences.

Self-attention networks with dynamic computational cost. DeeBERT (Xin et al., 2020), FastBERT (Liu et al., 2020), and Schwartz et al. (2020) each describe a method to dynamically adjust the amount of computation for different sequences. The intuition is that some sequences are easier to classify than others, and the "easy" sequences can be correctly classified by only computing the first few layers of a BERT-like network.

Convolutions in self-attention networks for language-generation tasks. In this paper, our experiments focus on natural language understanding (NLU) tasks such as sentence classification. However, another widely-studied area is natural language generation (NLG), which includes the tasks of machine-translation (e.g., English-to-German) and language modeling (e.g., automated sentence-completion). While we are not aware of work that adopts convolutions in self-attention networks for NLU, we *are* aware of such work in NLG. For instance, the Evolved Transformer and Lite Transformer architectures contain self-attention modules and convolutions in separate portions of the network (So et al., 2019; Wu et al., 2020). Additionally, LightConv shows that well-designed convolutional networks without self-attention produce comparable results to self-attention networks on certain NLG tasks (Wu et al., 2019b). Also, Wang *et al.* sparsify the self-attention matrix multiplication using a pattern of nonzeros that is inspired by dilated convolutions (Wang et al., 2020a). Finally, while not an attention network, Kim applied convolutional networks to NLU several years before the development of multi-head self-attention (Kim, 2014).

7 Conclusions & Future Work

In this paper, we have studied how grouped convolutions, a popular technique in the design of efficient computer vision neural networks, can be applied to natural language processing. First, we showed that the position-wise fully-connected layers of self-attention networks can be implemented with mathematically-equivalent 1D convolutions. Further, we proposed SqueezeBERT, an efficient NLP model which implements most of the layers of its self-attention encoder with 1D grouped convolutions. This model yields an appreciable >4x latency decrease over BERT-base when benchmarked on a Pixel 3 phone. We also successfully applied distillation to improve our approach's accuracy to a level that is competitive with a distillation-trained MobileBERT and with the original version of BERT-base.

We now discuss some possibilities for future work in the design of computationally-efficient neural networks for NLP. As we observed in Section 6, in recent months numerous approaches have been proposed for reducing the computational cost of self-attention neural architectures for natural language processing. These approaches include new model structures (e.g. MobileBERT), rethinking the dimensions of attention calculations (e.g. Linformer), grouped convolutions (SqueezeBERT), and much more. Further, once the neural architecture has been selected, approaches such as quantization and pruning can further reduce some of the costs associated with self-attention neural network inference. The combination of all of these potential techniques opens up a broad search-space of neural architecture designs for NLP. This motivates the application of automated neural architecture search (NAS) approaches such as those described in (Shaw et al., 2019; Wu et al., 2019a) to further improve the design of neural networks for NLP.

Acknowledgements

K. Keutzer's research is supported by Alibaba, Amazon, Google, Facebook, Intel, and Samsung. We would like to thank the EMNLP SustaiNLP Workshop's reviewers for their helpful comments on this paper.

References

Abrar Al-Heeti. 2018. WhatsApp: 65B messages sent each day, and more than 2B minutes of calls. *CNET*.

Iz Beltagy, Matthew E. Peters, and Arman Cohan. 2020. Longformer: The long-document transformer. *arXiv:2004.05150*.

Luisa Bentivogli, Ido Dagan, Hoa Trang Dang, Danilo Giampiccolo, and Bernardo Magnini. 2009. The fifth pascal recognizing textual entailment challenge. In *Text Analysis Conference (TAC)*.

Daniel Cer, Mona Diab, Eneko Agirre, Inigo Lopez-Gazpio, and Lucia Specia. 2017. Semeval-2017 task 1: Semantic textual similarity-multilingual and cross-lingual focused evaluation. In *Eleventh International Workshop on Semantic Evaluations*.

Zihan Chen, Hongbo Zhang, Xiaoji Zhang, and Leqi Zhao. 2018. Quora question pairs.

Francois Chollet. 2016. Xception: Deep learning with depthwise separable convolutions. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*. https://arxiv.org/abs/1610.02357.

Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training text encoders as discriminators rather than generators. In *International Conference on Learning Representations (ICLR)*.

Zihang Dai, Guokun Lai, Yiming Yang, and Quoc V. Le. 2020. Funnel-transformer: Filtering out sequential redundancy for efficient language processing. *arXiv:2006.03236*.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

William B. Dolan and Chris Brockett. 2005. Automatically constructing a corpus of sentential paraphrases. In *Proceedings of the International Workshop on Paraphrasing*.

Gordon Donnelly. 2018. 75 super-useful facebook statistics for 2018. https://www.wordstream.com/blog/ws/2017/11/07/facebook-statistics.

Kunihiko Fukushima. 1980. Neocognitron: A self-organizing neural network model for a mechanism of pattern recognition unaffected by shift in position. *Biological Cybernetics*.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. 2016. Deep residual learning for image recognition. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. 2015. Distilling the knowledge in a neural network. *arXiv:1503.02531*.

Andrew G. Howard, Menglong Zhu, Bo Chen, Dmitry Kalenichenko, Weijun Wang, Tobias Weyand, Marco Andreetto, and Hartwig Adam. 2017. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*.

Forrest Iandola and Kurt Keutzer. 2017. Small neural nets are beautiful: Enabling embedded systems with small deep-neural-network architectures. In *ESWEEK Keynote*.

Forrest N. Iandola, Khalid Ashraf, Matthew W. Moskewicz, and Kurt Keutzer. 2016a. FireCaffe: near-linear acceleration of deep neural network training on compute clusters. In *CVPR*.

Forrest N. Iandola, Song Han, Matthew W. Moskewicz, Khalid Ashraf, William J. Dally, and Kurt Keutzer. 2016b. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and <0.5mb model size. *arXiv:1602.07360*.

Xiaoqi Jiao, Yichun Yin, Lifeng Shang, Xin Jiang, Xiao Chen, Linlin Li, Fang Wang, and Qun Liu. 2019. TinyBERT: Distilling bert for natural language understanding. *arXiv:1909.10351*.

Nitish Shirish Keskar, Bryan McCann, Lav R. Varshney, Caiming Xiong, and Richard Socher. 2019. Ctrl: A conditional transformer language model for controllable generation. *arXiv:1909.05858*.

Yoon Kim. 2014. Convolutional neural networks for sentence classification. In *Conference on Empirical Methods in Natural Language Processing (EMNLP)*.

Vid Kocijan, Ana-Maria Cretu, Oana-Maria Camburu, Yordan Yordanov, and Thomas Lukasiewicz. 2019. A surprisingly robust trick for winograd schema challenge. In *ACL*.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton. 2012. ImageNet Classification with Deep Convolutional Neural Networks. In *NeurIPS*.

Alex Krizhevsky et al. 2011. cuda-convnet. https://code.google.com/archive/p/cuda-convnet.

Zhenzhong Lan, Mingda Chen, Sebastian Goodman, Kevin Gimpel, Piyush Sharma, and Radu Soricut. 2019. ALBERT: A lite bert for self-supervised learning of language representations. In *ICLR*.

Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel. 1989. Backpropagation applied to handwritten zip code recognition. *Neural Computation*.

Hector J Levesque, Ernest Davis, and Leora Morgenstern. 2012. The winograd schema challenge. In *Proceedings of the Thirteenth International Conference on the Principles of Knowledge Representation and Reasoning*.

Weijie Liu, Peng Zhou, Zhe Zhao, Zhiruo Wang, Haotang Deng, and Qi Ju. 2020. Fastbert: a self-distilling bert with adaptive inference time. *arXiv:2004.02178*.

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:1907.11692*.

Ilya Loshchilov and Frank Hutter. 2019. Decoupled weight decay regularization. In *ICLR*.

Lovely Mobile News. 2017. Mobile has largely displaced other channels for email.

NVIDIA. 2020a. APEX - A PyTorch Extension: Tools for easy mixed precision and distributed training in pytorch. `https://github.com/NVIDIA/apex`.

NVIDIA. 2020b. Deep learning examples for tensor cores. `https://github.com/NVIDIA/DeepLearningExamples`.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Köpf, Edward Yang, Zach DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. 2019. PyTorch: An imperative style, high-performance deep learning library. In *NeurIPS*.

Jason Phang, Thibault Févry, and Samuel R. Bowman. 2018. Sentence encoders on stilts: Supplementary training on intermediate labeled-data tasks. *arXiv:1811.01088*.

Quora. 2017. Quora question pairs.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. 2018. Improving language understanding by generative pre-training. `https://s3-us-west-2.amazonaws.com/openai-assets/research-covers/language-unsupervised/language_understanding_paper.pdf`.

Alec Radford, Jeff Wu, Rewon Child, David Luan, Dario Amodei, and Ilya Sutskever. 2019. Language models are unsupervised multitask learners. `https://d4mucfpksywv.cloudfront.net/better-language-models/language_models_are_unsupervised_multitask_learners.pdf`.

Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *International Conference on Medical image computing and computer-assisted intervention*.

Olga Russakovsky, Jia Deng, Hao Su, Jonathan Krause, Sanjeev Satheesh, Sean Ma, Zhiheng Huang, Andrej Karpathy, Aditya Khosla, Michael Bernstein, Alexander C. Berg, and Li Fei-Fei. 2015. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision*.

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

Victor Sanh, Thomas Wolf, and Alexander M. Rush. 2020. Movement pruning: Adaptive sparsity by fine-tuning. *arXiv:2005.07683*.

David Sayce. 2019. The number of tweets per day in 2019. `https://www.dsayce.com/social-media/tweets-day/`.

Jeff Schultz. 2019. How much data is created on the internet each day?

Mike Schuster and Kaisuke Nakajima. 2012. Japanese and korean voice search. In *International Conference on Acoustics, Speech and Signal Processing (ICASSP)*.

Roy Schwartz, Gabriel Stanovsky, Swabha Swayamdipta, Jesse Dodge, and Noah A. Smith. 2020. The right tool for the job: Matching model and instance complexities. *arXiv:2004.07453*.

Albert Shaw, Daniel Hunter, Forrest Iandola, and Sammy Sidhu. 2019. SqueezeNAS: Fast neural architecture search for faster semantic segmentation. In *ICCV Neural Architects Workshop*.

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 *AAAI*.

David R. So, Chen Liang, and Quoc V. Le. 2019. The evolved transformer. In *ICLR*.

Richard Socher, Alex Perelygin, Jean Wu, Jason Chuang, Christopher D. Manning, Andrew Ng, and Christopher Potts. 2013. Recursive deep models for semantic compositionality over a sentiment treebank. In *EMNLP*.

Siqi Sun, Yu Cheng, Zhe Gan, and Jingjing Liu. 2019a. Patient knowledge distillation for BERT model compression. In *Conference on Empirical Methods in Natural Language Processing and the International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*.

Zhiqing Sun, Hongkun Yu, Xiaodan Song, Renjie Liu, Yiming Yang, and Denny Zhou. 2019b. MobileBERT: Task-agnostic compression of BERT by progressive knowledge transfer. *OpenReview submission*.

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 *Annual Meeting of the Association for Computational Linguistics (ACL)*. ArXiv:2004.02984.

Mingxing Tan and Quoc V. Le. 2019. EfficientNet: Rethinking model scaling for convolutional neural networks. In *International Conference on Machine Learning (ICML)*.

Yi Tay, Mostafa Dehghani, Dara Bahri, and Donald Metzler. 2020. Efficient transformers: A survey. *arXiv:2009.06732*.

Templatify. 2017. How many emails are sent every day? top email statistics for business.

Iulia Turc, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. Well-read students learn better: On the importance of pre-training compact models. *arXiv:1908.08962*.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Conference on Neural Information Processing Systems (NeurIPS)*.

Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2018. GLUE: A multi-task benchmark and analysis platform for natural language understanding. *arXiv:1804.07461*.

Chenguang Wang, Zihao Ye, Aston Zhang, Zheng Zhang, and Alexander J. Smola. 2020a. Transformer on a diet. *arXiv:2002.06170*.

Sinong Wang, Belinda Z. Li, Madian Khabsa, Han Fang, and Hao Ma. 2020b. Linformer: Self-attention with linear complexity. *arXiv:2006.04768*.

Alex Warstadt, Amanpreet Singh, and Samuel R. Bowman. 2019. Neural network acceptability judgments. In *Transactions of the Association for Computational Linguistics*.

Adina Williams, Nikita Nangia, and Samuel R. Bowman. 2018. A broad-coverage challenge corpus for sentence understanding through inference. In *Conference of the North American Chapter of the Association for Computational Linguistics (NAACL)*.

Thomas Wolf, Lysandre Debut, Victor Sanh, Julien Chaumond, Clement Delangue, Anthony Moi, Pierric Cistac, Tim Rault, Rémi Louf, Morgan Funtowicz, and Jamie Brew. 2019. Huggingface's transformers: State-of-the-art natural language processing. *arXiv:1910.03771*.

Bichen Wu, Xiaoliang Dai, Peizhao Zhang, Yanghan Wang, Fei Sun, Yiming Wu, Yuandong Tian, Peter Vajda, Yangqing Jia, and Kurt Keutzer. 2019a. FBNet: Hardware-aware efficient convnet design via differentiable neural architecture search. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Felix Wu, Angela Fan, Alexei Baevski, Yann N. Dauphin, and Michael Auli. 2019b. Pay less attention with lightweight and dynamic convolutions. In *ICLR*.

Yonghui Wu, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Łukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes, and Jeffrey Dean. 2016. Google's neural machine translation system: Bridging the gap between human and machine translation. *arXiv:1609.08144*.

Zhanghao Wu, Zhijian Liu, Ji Lin, Yujun Lin, and Song Han. 2020. Lite transformer with long short term attention. In *ICLR*.

Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. 2017. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*.

Ji Xin, Raphael Tang, Jaejun Lee, Yaoliang Yu, and Jimmy Lin. 2020. Deebert: Dynamic early exiting for accelerating bert inference. In *ACL*.

Canwen Xu, Wangchunshu Zhou, Tao Ge, Furu Wei, and Ming Zhou. 2020. Bert-of-theseus: Compressing bert by progressive module replacing. *arXiv:2002.02925*.

Yang You, Jing Li, Sashank Reddi, Jonathan Hseu, Sanjiv Kumar, Srinadh Bhojanapalli, Xiaodan Song, James Demmel, Kurt Keutzer, and Cho-Jui Hsieh. 2020. Large batch optimization for deep learning: Training bert in 76 minutes. In *ICLR*.

Ofir Zafrir, Guy Boudoukh, Peter Izsak, and Moshe Wasserblat. 2019. Q8BERT: Quantized 8bit bert. *arXiv:1910.06188*.

Xiangyu Zhang, Xinyu Zhou, Mengxiao Lin, and Jian Sun. 2018. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. In *CVPR*.

Yukun Zhu, Ryan Kiros, Rich Zemel, Ruslan Salakhutdinov, Raquel Urtasun, Antonio Torralba, and Sanja Fidler. 2015. Aligning books and movies: Towards story-like visual explanations by watching movies and reading books. In *IEEE International Conference on Computer Vision (ICCV)*.