Layer-Wise Neural Network Compression via Layer Fusion

James O'Neill JAMES.O-NEILL@LIVERPOOL.AC.UK
Department of Computer Science, University of Liverpool, Liverpool, England

Greg V. Steeg GREGV@ISI.EDU
Aram Galstyan GALSTYAN@ISI.EDU
USC Information Sciences Institute, Marina del Rey, California, 90292, USA

Editors: Vineeth N Balasubramanian and Ivor Tsang

Abstract

This paper proposes *layer fusion* - a model compression technique that discovers which weights to combine and then fuses weights of similar fully-connected, convolutional and attention layers. Layer fusion can significantly reduce the number of layers of the original network with little additional computation overhead, while maintaining competitive performance. From experiments on CIFAR-10, we find that various deep convolution neural networks can remain within 2% accuracy points of the original networks up to a compression ratio of 3.33 when iteratively retrained with layer fusion. For experiments on the WikiText-2 language modelling dataset, we compress Transformer models to 20% of their original size while being within 5 perplexity points of the original network. We also find that other well-established compression techniques can achieve competitive performance when compared to their original networks given a sufficient number of retraining steps. Generally, we observe a clear inflection point in performance as the amount of compression increases, suggesting a bound on the amount of compression that can be achieved before an exponential degradation in performance.

Keywords: Weight Sharing, Model Compression, Language Modeling, Image Classification

1. Introduction

Deep neural networks (DNNs) have made a significant impact on fields such as Computer Vision (CV) (He et al., 2016; Iandola et al., 2014) and Natural Language Processing (NLP) (Vaswani et al., 2017; Devlin et al., 2018). Deep Convolutional Neural Networks (CNNs) (Krizhevsky et al., 2012) have improved performance on image classification (Krizhevsky et al., 2012), image segmentation (Long et al., 2015), speech recognition (LeCun et al., 1995) and have been widely adopted in the machine learning (ML) community. This has been accelerated due to numerous innovations such as skip connections in ResNets (He et al., 2016) to avoid the vanishing gradient problem and batch normalization (Ioffe and Szegedy, 2015; Ba et al., 2016) and layer normalization (Ba et al., 2016) to reduce the effects of shifts in the training and test data distributions. Similarly, Transformer models have shown great success in NLP due to the use of self-attention (Vaswani et al., 2017), significantly outperforming preceding Recurrent Neural Network (RNN) based architectures (Hochreiter and Schmidhuber, 1997) on a diverse set of NLP tasks (Radford et al., 2018; Dehghani et al., 2018; Dai et al., 2019). However, these large overparameterized networks require more compute, training time, storage and leave a larger carbon footprint (Strubell et al., 2019). While prior work on model compression has mainly focused on deploying compressed models to mobile devices (Han et al., 2015a; Wu et al., 2016), moving models from multi-GPU training to single-GPU training is now too

a salient challenge. If achieved, this relaxes the resource requirements for ML practitioners and allow a wider adoption of larger pretrained CNNs and Transformers.

This leads us to ask the following questions on DNNs: *are all layers of large pretrained models required for a given target task ? If not, can we reduce the network while preserving network density (i.e no non-zero parameters) during retraining in a computationally efficient manner ?*. Earlier work (He et al., 2016) on CNNs found that some layers may become redundant in very deep networks, essentially copying earlier layers and performing identity mappings for the redundant layers. While residual connections have ameliorated these problems to some degree (not only in residual networks e.g Transformers), we assert that there may still be significant redundancy between layers of large overparameterized networks. More recently, Zhang et al. (2019) found that whole layers can be distinctly separated by their importance in prediction, further motivating us to seek a compression technique that identifies and uses salient layers. However, many current compression techniques are not equipped to preserve these salient layers during model compression of pretrained models because they are unstructured techniques (Karnin, 1990; Hassibi and Stork, 1993; Han et al., 2015b), resulting in a sparse model. This is a practical limitation since sparse networks require more conditional operations to represent which elements within each parameter matrix are zero or non-zero. Current GPU libraries such as CuSPARSE (accounting for recent improvements (Argueta and Chiang, 2019)) are far slower than CuBLAS (Sanders and Kandrot, 2010) and current hardware is not designed to optimize for sparse matrix operations.

In contrast, knowledge distillation (Hinton et al., 2015; Mishra and Marr, 2017; Ashok et al., 2017) and quantization (Polino et al., 2018) preserve network density, avoiding the necessity for specialized sparse matrix libraries to utilize the benefits of smaller and faster networks. However, quantization leads to quantization error and requires approximate methods to compute partial derivatives during retraining (Agustsson et al., 2017) and knowledge distillation requires more memory to store and train the smaller network. Weight sharing reduces the network size and avoids sparsity, however it is unclear which weights should be shared and it cannot be used when the model is already pretrained with untied weights. The noted drawbacks of the aforementioned compression methods further motivates us to seek an alternative structured compression method that preserves network density while identifying and removing redundant layers. This brings us to our main contributions.

Contributions We propose *layer fusion* (LF). LF aims to preserve information across layers during retraining of already learned models while preserving layer density for computational efficiency. We also propose *alignment* measures for LF, since aligning paths, layers and whole neural networks is non-trivial (neurons can be permuted and still exhibit similar behaviour and we desire an invariance to orthogonal transformations). This includes (1) a Wasserstein distance metric to approximate the alignment cost between weight matrices and (2) numerous criteria for measuring similarity between weight covariance matrices. We use these measures as LF criteria to rank layer pairs that are subsequently used to fuse convolutional, fully-connected and attention layers. This leads to both computational and performance improvements over layer removal, pruning and shows competitive results compared to tensor-decomposition and unsupervised knowledge distillation. We report results of LF using different fusion approaches: layer freezing, averaging and random *mixing*. We use current structured compression techniques as baselines for both CNNs and Transformers, with and without retraining and identify a reoccurring inflection point w.r.t performance versus model size.

2. Related Work

Layer Structure & Importance Zhang et al. (2019) have recently analysed the layer-wise functional structure of overparameterized deep models to gain insight into why deep networks have performance advantages over their shallow counterparts. They find that some layers are salient and that once removed, or reinitialized, have a catastrophic effect on learning during training and subsequently generalization. In contrast, the remaining layers once reset to their default initialization has little effect. This suggests that parameter and norm counting is too broad of a measure to succinctly study the generalization properties in deep networks. These findings also motivate LF, as we posit that important layers are more distinct and therefore will be less similar, or harder to align with other layers, while more redundant layers may be good candidates for fusing layers. Frankle and Carbin (2018) showed that there exists trained subnetworks that when re-initialized to their original configuration produce the same performance as the original network in the same number of training epochs. They also posit that stochastic gradient descent (SGD) seeks out a set of *lottery tickets* (i.e well-initialized weights that make up a subnetwork that when trained for the same number of epochs as the original network, or less, can reach the same out-of-sample performance) and essentially ignores the remaining weights. We can further conjecture from Zhang et al. (2019) findings, that perhaps SGD more generally seeks out important layers, which we analogously refer to as *lottery pools*. Identifying whole layers that are significantly distinguished from others, in terms of their influence on learning, further motivates us to merge or freeze layers.

Computing Layer Similarity Kornblith et al. (2019) have focused on computing similarity between different neural representations (i.e the activation output vector for a given layer). However, we view this comparison between layers as slightly limiting, since information is lost about what weights and bias led to the activation outputs. Moreover, directly comparing neural network weights allows us to avoid sampling inputs to compute the activations. In contrast, work focusing on representational similarity across networks Li et al. (2016); Kornblith et al. (2019), we are instead comparing weight matrices within the same network. Directly comparing weights and biases allow us to better approximate alignments and similarities for dense networks and has the advantage that we do not require data to be feed-forward through the network post-training to measure similarity within or across networks, unlike representational similarity (i.e output activations).

Structured Dropout Fan et al. (2019) proposes to randomly drop whole layers during training time and at test time they can choose a subnetwork which can be decided based on performance of different combinations of pruned networks on the validation set or based on dropout probabilities learned for each layer throughout training. Singh and Jaggi (2019) measure model similarity across neural networks using optimal transport-based metrics. In contrast, our work measures intra-network similarity and we make a distributional assumption that allows us to use such metric efficiently during retraining, making it feasible to scale for large networks.

3. Methodology

We begin by defining a dataset as $\{\mathcal{D} = (\boldsymbol{x}_i, \boldsymbol{y}_i) : i = 1, \dots T\}$ that contains T tuples of an input vector $\boldsymbol{x} \in \mathbb{R}^n$ and a corresponding target $\boldsymbol{y} \in \{0, 1\}^p$. We define any arbitary sample as $s := (\boldsymbol{x}, \boldsymbol{y})$ where $s \in \mathcal{D}$. We consider a neural network $f_\theta(\boldsymbol{x})$ with pretrained parameters $\theta := (\theta_1, \theta_2, \dots \theta_\ell \dots, \theta_L)^T$. Here $\theta_\ell := \{\mathbf{W}_\ell, \boldsymbol{b}_\ell\}$ where $\mathbf{W}_\ell \in \mathbb{R}^{n_\ell \times n_{\ell+1}}$, $\boldsymbol{b} \in \mathbb{R}^{n_{\ell+1}}$ where n_ℓ denotes the dimension size of the ℓ-th layer. Thus, a standard fully-connected f_θ is expressed as,

$$f_\theta(\boldsymbol{x}) := \mathbf{W}_L g\Big(\dots g\big(\mathbf{W}_2 g(\mathbf{W}_1 \boldsymbol{x} + \boldsymbol{b}_1) + \boldsymbol{b}_2\big) + \boldsymbol{b}_L \tag{1}$$

with smooth asymptotic nonlinear function $g(\cdot)$ (e.g hyperbolic tangent) that performs element-wise operations on its input. The input to each subsequent layer as $\boldsymbol{z}_\ell \in \mathbb{R}^{n_\ell}$ where $\boldsymbol{x} := \boldsymbol{z}_0$ for m number of units in layer ℓ and the corresponding output activation as $\boldsymbol{T}_\ell = g(\boldsymbol{z}_\ell)$. The loss function is defined as $\mathbb{L}_\theta(\mathcal{D}) := \frac{1}{T}\sum_{i=1}^{N} \mathcal{L}(\boldsymbol{y}_i, f_\theta(\boldsymbol{x}_i))$ where for a single sample s_i, $\mathcal{L} : \mathcal{Y} \times \mathbb{R}^n \to \mathbb{R}$. A pruned θ_ℓ post-training is denoted as θ_ℓ^p and a tensor decomposed θ_ℓ is expressed as $\tilde{\theta}_\ell$ where $\tilde{\mathbf{W}}_\ell \in \mathbb{R}^{d_\ell \times d_{\ell+1}}$ and $\tilde{\boldsymbol{b}}_\ell \in \mathbb{R}^{d_{\ell+1}}$ and $d \ll n$. A network pruned by layer as a percentage of the lowest weight magnitudes is denoted as $f_{\acute{\theta}}^{lp}$ where the pruned weights $\acute{\theta} \subset \theta$. A percentage of the network pruned by weight magnitudes across the whole network is denoted as $f_{\acute{\theta}}^{gp}$ (i.e global pruning). Lastly, a post layer fused network f_Θ has fused parameters Θ.

3.1. Desirable Properties of Weight Similarity

Ideally, we seek a measure that can compare weight matrices that are permutable and of varying length. Formally, the main challenges with aligning weight matrices $\mathbf{W} := \{\mathbf{W}_0, \dots, \mathbf{W}_\ell, \dots \mathbf{W}_L\}$ of different layers is that, when vectorized as $\texttt{vec}(\mathbf{W}_\ell) \in \mathbb{R}^{n_\ell(n_{\ell+1})}$, $\mathbf{W}_\ell$ can be permuted and still exhibit the same behavior at the output. Hence, if $|\mathbf{W}_\ell| \neq |\mathbf{W}_{\ell+1}|$, the measure S must allow for multisets of different lengths and permutations. Invariance to rotations, reflections and scaling are all desirable properties we aim to incorporate into measuring similarity between weight matrices. However, invariance to linear transformations has issues when there are more parameters in a layer than training samples, as pointed out by Kornblith et al. (2019). Even though our work mainly focuses on large pretrained models, we also seek a LF measure that is invariant to orthogonal transformations to overcome the aforementioned issues i.e for a similarity function $s(\cdot,\cdot)$, $s(\mathbf{W}_i, \mathbf{W}_j) = s(\mathbf{W}_i\mathbf{U}, \mathbf{W}_j\mathbf{V})$ for full-rank orthonormal matrices $\mathbf{U}$ and $\mathbf{V}$ such that $\mathbf{U}^T\mathbf{U} = \mathbf{I}$ and $\mathbf{V}^T\mathbf{V} = \mathbf{I}$. More importantly, invariance to orthogonal transformation relates to permutation invariance (Orhan and Pitkow, 2017) which is a property we account for when measuring the alignment between weight matrices. Lastly, we note that if two weight sets are of unequal size we randomly downsample the larger weight set to match the paired weight set. This is required for aligning filters in CNN networks after vectorization. We now describe a set of measures we consider for aligning and measuring the similarity of layers.

3.2. Layer Alignment & Layer Similarity

Covariance Alignment The first layer fusion measure we consider is covariance alignment (CA). CA accounts for correlated intra-variant distances between layers, which can indicate some redundancy, although their overall distributions may differ and therefore may be good candidates for LF. Hence, we consider the Frobenius norm (denoted as subscript F) between pairs of weight covariance matrices $\mathbf{\Sigma}_{\tilde{\mathbf{W}}_1}, \mathbf{\Sigma}_{\tilde{\mathbf{W}}_2}$ and expectation $\mathbb{E}[\mathbf{W}_1] = \mathbb{E}[\mathbf{W}_2] = 0$. This forms a Riemannian manifold of non-positive curvature over the weight covariances. We first consider the cosine distance as the distance measures between parameter covariance matrices as Equation 2, where $||\Sigma_\mathbf{W}||_F = [\text{tr}(\Sigma_\mathbf{W}^T\Sigma_\mathbf{W})]^{1/2}$.

$$D_{\cos}(\mathbf{\Sigma}_{\mathbf{W}_1}, \mathbf{\Sigma}_{\mathbf{W}_2}) = \frac{\text{tr}\left(\mathbf{\Sigma}_{\mathbf{W}_1} \cdot \mathbf{\Sigma}_{\mathbf{W}_2}\right)}{||\mathbf{\Sigma}_{\mathbf{W}_1}||_F ||\mathbf{\Sigma}_{\mathbf{W}_2}||_F} \tag{2}$$

If we assume both weight matrices are drawn from a normal distribution $\mathbf{W}_1 \sim \mathcal{N}(\mu, \sigma_1), \mathbf{W}_2 \sim \mathcal{N}(\mu, \sigma_2)$ with identical means $\mu = \mu_{\mathbf{W}_1} = \mu_{\mathbf{W}_2}$, the KL divergence between their covariance matrices can be expressed as:

$$D_{\mathrm{KL}}(\mathbf{\Sigma}_{\mathbf{W}_1} || \mathbf{\Sigma}_{\mathbf{W}_2}) = \frac{1}{2}\big[\,\mathrm{tr}\,\big(\mathbf{\Sigma}_{\mathbf{W}_2}^{-1}\mathbf{\Sigma}_{\mathbf{W}_1}\big) - \ln\big(\frac{|\mathbf{\Sigma}_{\mathbf{W}_1}|}{|\mathbf{\Sigma}_{\mathbf{W}_2}|}\big)\big] \tag{3}$$

The symmetrized KL divergence between positive semi-definite matrices (e.g covariances) also acts as the square of a distance Moakher and Batchelor (2006) (see the supplementary for further details, including descriptions of other covariance similarity measures). We consider both Equation 2 and Equation 3 for fusing convolutional layers, self-attention layers and fully-connected layers. The KL is an asymmetric measure, therefore the divergence in both directions can be used to assign a weight to each layer pair in layer fusion.

Optimal Transport & Wasserstein Distance Unlike an all-pair distance such as CA, Wasserstein (WS) distance can also be used to find the optimal cost, also known as the optimal flow between two distributions. Unlike, other distance measures, WD tries to keep the geometry of the distributions intact when interpolating and measuring between distributions. Unlike CA and other baseline measures, WS is invariant to layer permutations and like CA, it also accounts for mutual dependencies between parameters in any arbitrary layer. In this work, we consider the WD between adjacent row-normalized parameter pairs $\mathrm{softmax}(\mathbf{W}_1, \mathbf{W}_2)$ (i.e multisets) in a Euclidean metric space. Given two multi-sets $\mathbf{W}_1, \mathbf{W}_2 \subset \mathbf{W}$, of size $d = |\mathbf{W}_1| = |\mathbf{W}_2|$ with corresponding empirical distributions $P_{\mathbf{W}_1}$ and $P_{\mathbf{W}_2}$, the WS distance is defined as Equation 4. However, computing WS distance is $\mathcal{O}(N^3)$ using the standard Hungarian algorithm, which is intractable for large θ.

$$D_{W_p}(P_{\mathbf{W}_1}, P_{\mathbf{W}_2}) = \inf_{\pi}\Big(\sum_{i=1}^{d} ||P_{\mathbf{W}_1^i} - P_{\mathbf{W}_2^{\pi(i)}}||^p\Big)^{1/p} \tag{4}$$

One way to tradeoff this computational burden is to assume that the weights are i.i.d and normally distributed at the expense of disregarding mutual dependencies learned throughout training. According to Lyapunov's central limit theorem (Lehmann, 2004), we can assume the the weights in a layer are normally distributed. Hence, if $P_{\mathbf{W}_1} = N(\mu_{\mathbf{W}_1}, \mathbf{\Sigma}_1)$ and $P_{\mathbf{W}_2} = N(\mu_{\mathbf{W}_2}, \mathbf{\Sigma}_2)$ we can express the 2-WS distance as Equation 5, also known as the Bures metric[1].

$$\begin{aligned} D_{\mathbb{W}^2}(P_{\mathbf{W}_1}, P_{\mathbf{W}_2}) &= ||\mu_{\mathbf{W}_1} - \mu_{\mathbf{W}_2}||^2 + \mathbb{B}^2(\mathbf{\Sigma}_{\mathbf{W}_1}, \mathbf{\Sigma}_{\mathbf{W}_2}), \\ \mathbb{B}^2(\mathbf{\Sigma}_{\mathbf{W}_1}, \mathbf{\Sigma}_{\mathbf{W}_2}) &= \mathrm{tr}(\mathbf{\Sigma}_{\mathbf{W}_1}) + \mathrm{tr}(\mathbf{\Sigma}_{\mathbf{W}_2}) - \\ &\quad 2\mathrm{tr}\Big[\big(\sqrt{\mathbf{\Sigma}_{\mathbf{W}_1}}(\mathbf{\Sigma}_{\mathbf{W}_2}\sqrt{\mathbf{\Sigma}_{\mathbf{W}_1}})\big)\Big] \end{aligned} \tag{5}$$

Although we focus on the Bures metric in our experiments, an alternative approach is to find a set of cluster centroids in $\mathbf{W}_1$ and $\mathbf{W}_2$ as $C_{\mathbf{W}_2}$ and $C_{\mathbf{W}_2}$ and compute $W(P_{C_{\mathbf{W}_1}}, P_{C_{\mathbf{W}_2}})$. In this approach the centroids are converted to an empirical distribution P_{C_θ} such $c \ll d$ such that a $\mathcal{O}(N^3)$ cost is feasible for computing during retraining steps. Alternatively, we could avoid softmax normalization and directly compute $\mathbb{W}(C_{\mathbf{W}_1}, C_{\mathbf{W}_2})$ on both discrete sets. Lastly, we note that when fusing layers with 2-WS distance, the fusion occurs between aligned weights given by the cost matrix. Hence, it is not only used to identify top-k most similar layers, but the cost matrix also aligns which weights in the layer pair are fused.

1. Often used in quantum physics for measuring quantum state correlations (Forrester and Kieburg, 2016).

3.3. Fusing Layers

After choosing the top-k layer pairs to merge, we then consider 3 ways to fuse the layers: (1) freeze one of the two layers and freeze the gradients for one of the two layers, (2) take the mean between layer pairs and compute backprop on the averaged layer pair and (3) sample and mix between the layers. Merging layer pairs with (1), refers to lines 12-14 in Algorithm 1. Choosing the layers to fuse for (1) is based on which of the two is closest to the middle layer of the network. This is motivated by previous work that showed layers closer to the input and output are generally more salient (Zhang et al., 2019). When using Jenson-Shannon divergence for choosing top-k layers, we use the divergence asymmetry for choosing which layer is frozen. This is achieved by taking the parameter γ between the Jenson-Shannon divergence of two layers in both directions to control a weighted gradient. We express the backpropogation when using LF with Jenson-Shannon divergence in terms of KL-divergences as shown in Equation 6, where $\bar{\mathbf{W}}_{ij}$ is a mixture distribution between $\mathbf{W}_i$ and $\mathbf{W}_j$ with a weighted gradient $\partial\mathcal{L}/\partial\tilde{\mathbf{W}}_{ij}$ that represents the gradient for both $\mathbf{W}_i$ and $\mathbf{W}_j$. Thus, for the backward pass of a frozen layer from a given top-k pair, we still compute its gradients which will influence how its original pair will be updated. This constraint ensures that the original pair that were most similar for a given compression step remain relatively close throughout retraining. The layer pair are then averaged at test time to reduce network size, while maintaining similarity using the aforementioned JS divergence gradient constraint.

Algorithm 1 Layer Fusion algorithm. (LFA)

1: **input:**alignment function D, parameters θ, k layers to merge.
2: $\mathbf{S} = \mathbf{0}_{|\theta|,|\theta|}$
3: **for** $\mathbf{W}_i$ in $\{\boldsymbol{\theta}_i\}_{i=1}^{|\theta|}$ **do**
4: **for** $\mathbf{W}_j$ in $\{\boldsymbol{\theta}_i\}_{j=i}^{|\theta|}$ **do**
5: **if** $\text{param\_type}(\theta_i) == \text{param\_type}(\theta_j)$ **then**
6: $\mathbf{S}_{ij} = D(\text{vec}(\mathbf{W}_i), \text{vec}(\mathbf{W}_j))$
7: **end if**
8: **end for**
9: **end for**
10: $\{I\}$ list of tuples of matrix indices $\arg\max_k \mathbf{S}$

11: **for** $(i_1, \dots i_k)$ in $\{I\}$ **do**
12: $\tilde{\mathbf{W}}_{ij} \sim \mathbb{B}(\mathbf{W}_i, \mathbf{W}_j)$
13: $\tilde{\boldsymbol{b}}_{ij} \sim \mathbb{B}(\boldsymbol{b}_i, \boldsymbol{b}_j)$
14: $\theta_i = \theta_j = \{\tilde{\mathbf{W}}_i, \tilde{\boldsymbol{b}}_i\}$
15: **end for**
16: **return** θ

4. Experimental Details

We focus on Transformer-based models for language modelling on the WikiText-2 dataset (Merity et al., 2016). For large models in NLP such as BERT (Devlin et al., 2018), OpenAI-GPT, GPT2 (Radford et al., 2018) and Transformer-XL (Yang et al., 2019), we freeze or combine layer weights of each multi-attention head component and intermediate dense layers, dramatically reducing the respective number of layer and weights.

$$\frac{\partial\mathcal{L}}{\partial\tilde{\mathbf{W}}_{ij}} := \gamma\Big(\frac{\partial L}{\mathbf{W}_i}\Big) + \big(1-\gamma\big)\Big(\frac{\partial L}{\mathbf{W}_j}\Big) \quad s.t, \quad \gamma = \frac{1}{2}\Big(D_{\text{KL}}(\mathbf{W}_i||\bar{\mathbf{W}}_{ij}) + D_{\text{KL}}(\mathbf{W}_j||\bar{\mathbf{W}}_{ij})\Big) \tag{6}$$

For (2), updates during training when using , we constrain the gradients to be the average of both layers and then average the resulting layers at the end of retraining. For (3), we interpolate between hidden representations that are most similar, which can be viewed as a stochastic approach of JS divergence used in (1), to remove redundancy in the network. We denote a pair of randomly

mixed layers as $\tilde{\mathbf{W}}_\ell^i \sim \mathbb{B}\big(\mathbf{W}_\ell^i, \mathbf{W}_{\ell+1}\big) \quad \forall i \in n_\ell$. Note that we only mix between pairs of weight matrices, the bias terms are averaged $(\boldsymbol{b}_\ell + \boldsymbol{b}_\ell)/2$. We then compute backpropogation on $\tilde{\theta}_\ell^i$ instead of the original unmixed layer pair $(\theta_\ell^i, \tilde{\theta}_{\ell+1}^i)$ i.e mixing is carried out before the forward pass. We summarize the standard retraining procedure in Algorithm 2 where LFA performs similarity and merging from Algorithm 1.

For image classification on CIFAR10, we report results for ResNet, ResNet-ELU (Shah et al., 2016), Wide-ResNet (Zagoruyko and Komodakis, 2016) and DenseNet (Huang et al., 2017). We are particularly interested in ResNet architectures, or more generally, ones that also use skip connections. This is motivated by Veit et al. (2016) which found that deleting or permuting residual blocks can be carried out without much degradation in performance in a pretrained ResNet.

Algorithm 2 Retraining algorithm. (RA)

input: batch size M, number of batches N, compression epoch interval N_c, pretrained f_θ, alignment function D.
for an epoch $\{\boldsymbol{\epsilon}_i\}_{i=1}^N$ **do**
  for $(\mathbf{X}, \boldsymbol{y})$ sampled minibatch $\{\boldsymbol{B}_i\}_{i=1}^N$ **do**
    $\hat{\boldsymbol{y}} = f_\theta(\boldsymbol{X})$
    Update gradients on f_θ to minimize $\ell_{\text{CE}}(\hat{\boldsymbol{y}}, \boldsymbol{y})$
  end for
  if $\epsilon_i \bmod N_c$ **then**
    $\theta = \text{LFA}(\theta, D, k)$
  end if
end for
return $f_\Theta(\cdot)$

4.1. Compression Without Retraining

For magnitude-based pruning, we prune a percentage of the weights with the lowest magnitude. This is done in one of two ways: a percentage of weights pruned by layer (layer pruning), or a percentage of the network as whole (global pruning). For quantization, we use k-means whereby the number of clusters for a given layer is specified as a percentage of the original size of that layer (i.e number of parameters in the tensor). For tensor decomposition, we reduce the number of parameters by approximating layers with a lower rank using singular value decomposition (SVD). Specifically, we use randomized truncated SVD (Halko et al., 2011) where QR factorization on $\mathbf{W}_\ell$ such that $\mathbf{Q}_\ell^T \mathbf{W}_\ell = \mathbf{R}_\ell$ where $\mathbf{Q}_\ell$ are the orthonormal columns of $\mathbf{W}_\ell$. Randomized methods are used to approximate the range of θ_ℓ and reduce computation from $\mathcal{O}(\min(n_{\ell-1} n_\ell^2, n_{\ell-1}^2 n_\ell))$ to $\mathcal{O}(n_{\ell-1} n_\ell \log(k))$ where k represents the approximate rank of θ_ℓ. We also perform dimensionality reduction on the layers by using 1-hidden layer denoising autoencoders which use the same activation functions for reconstruction as the original architecture and a mean squared error loss is minimized. The encoder layer of each denoising AE (DAE) is the used in replacement of the original layer. For both truncated SVD and DAE, this is carried out sequentially from bottom to top layer so that the reconstruction of a given layer l also accounts for cascade approximation errors of dimensionality reduction from previous layers. We refer to this type of layer reconstruction technique as *student rollout* because the pretrained teacher network is iteratively rolled out and reconstructed from the first layer to the last.

4.2. Layer Fusion & Compression ReTraining

For retraining we consider two main schemes: (1) for each retraining step we carry out network compression (e.g via pruning), retrain the resulting network and iteratively repeat until the final epoch, and (2) in the case where network compression leads to non-zero weights (e.g LF), we freeze

the network weights apart from those which have been identified for LF in which case we retrain before tying.

Layer averaging, mixing and freezing are experimented with for fusing layers. To maintain uniformity across each compression step, we prune, quantize, fuse and decompose a percentage of the weights as opposed to using other criteria such as the weight magnitude thresholding.

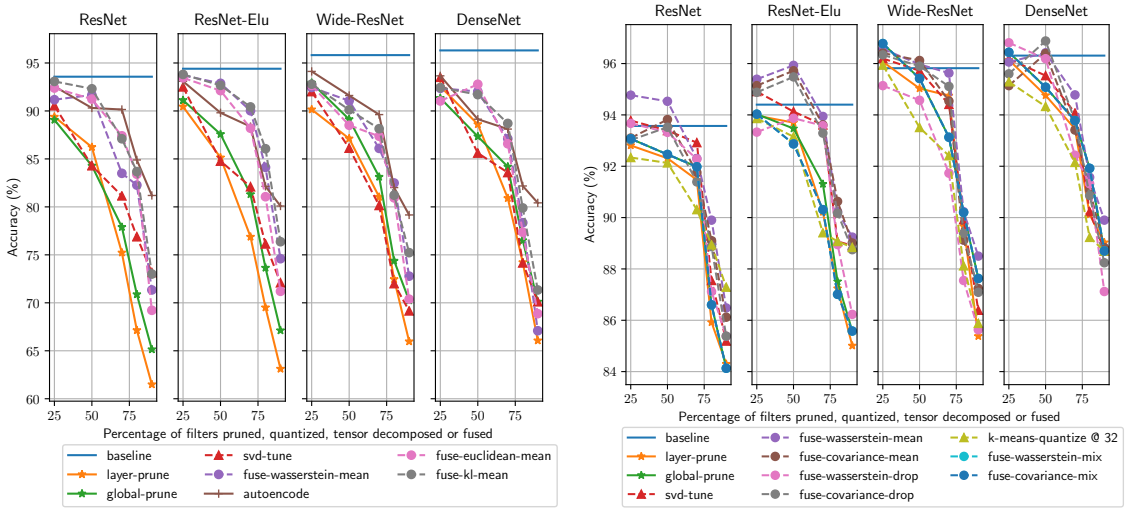
This ensures comparability between compression methods e.g thresholding weights in pruning does not have a direct equivalent to quantization or weight decomposition, unless network reduction is proportional to the number of weights pruned via thresholding.

5. Results

5.1. Image Classification

Figure 1(*a*)subfigure shows the results of pruning, quantization, weight decomposition and our proposed LF *without* any retraining. A general observation is that an exponential decline in performance occurs at around 70% (some variance depending on the compression method) of the original network is compressed. For example, fusing layers using the WS distance for alignment allows accuracy to be closer to the original networks accuracy up to 70%. In contrast, pruning convolutional layers in ResNet models leads to a faster accuracy drop. This is surprising given that unstructured pruning is less restrictive, when applying LF to CNN architectures. We also allow filters from the same layer to be fused, in comparison to dense layers in self-attention blocks for Transformers.

Figure 1(*b*)subfigure shows results of the compression techniques *with* retraining. We see the results of model compression methods retraining on CIFAR-10 for ResNet-50, ResNet-50 with exponential linear units (ELUs), Wide-ResNet and DenseNet. We test each combination of layer pairs for averaging layers as $\tilde{\theta}_i = \tilde{\theta}_j = (\theta_i + \theta_j/2)$ where $\binom{L}{2}$ are the total number of layers (e.g 24 layers results in 276 possible pairs). The performance change is measured from the original network when layer averaging by choosing the top $L \times \%$ and measuring which averaged layer pair



(*a*) No Retraining (*b*) Retraining

Figure 1: **CIFAR-10 Test Accuracy With And Without Retraining**

Table 1: **CIFAR-10 Test Accuracy with WS-Based CNN Layer Fusion**

		RES		RES-ELU		WIDE-RES		DENSENET	
ORIG.		93.75	-	94.40	-	95.82	-	96.31	-
MEAN	25%	92.39	94.77	93.45	95.39	92.66	96.57	91.04	96.06
	50%	91.24	94.53	92.12	95.93	88.51	95.97	92.78	96.42
	75%	87.41	92.30	88.20	93.94	87.36	95.63	86.58	94.78
	80%	83.40	89.90	81.06	90.23	80.94	90.23	77.36	88.50
	90%	**69.22**	**86.48**	**71.20**	**89.24**	**70.38**	**89.90**	**68.87**	**91.57**
FREEZE	25%	91.17	93.67	93.71	93.33	92.45	95.14	92.36	96.81
	50%	91.67	93.32	92.88	93.87	91.06	94.57	92.03	96.19
	75%	83.50	92.28	90.02	93.58	86.10	91.73	87.11	92.43
	80%	82.27	87.12	84.12	88.95	82.49	87.55	78.35	85.63
	90%	**71.34**	**85.38**	**74.60**	**86.23**	**72.78**	**85.63**	**67.09**	**87.12**
MIXING	25%	93.67	93.22	93.33	94.03	95.14	96.78	96.81	96.44
	50%	93.32	92.46	93.24	93.87	94.57	95.42	96.19	95.08
	70%	92.28	91.98	90.31	93.58	91.73	93.13	92.43	93.78
	80%	84.12	90.60	85.58	88.95	87.55	90.20	91.32	91.92
	90%	**73.2**	**86.13**	**73.50**	**85.58**	**80.63**	**87.63**	**78.12**	**88.70**

produced the smallest difference in accuracy when compared to the original network. When the same layer within the top $L \times \%$ is coupled with more than one other layer, we take the mean of multiple pairs to reduce computation to $2\binom{L}{L\times\%}$. We find a reoccurring pattern that early on retraining, we observe up to a reduction of 25% of the network improves the results, and even up to 25% - 50% in some cases (e.g global pruning and layer pruning). From 75% we see a significant decrease in performance, typically 2-4% drop in accuracy percentage points across each model. Given an allowance of N retraining epochs, allocating the amount of model compression for each compression step is a critical hyperparameter. Concretely, less retraining time is necessary for during initial model compression, whereas past a compression ratio of 3.33 (i.e 75%), the interval between retraining steps should become larger. This is highlighted in bold across Table 1, where left subcolumns are with no retraining and right subcolumns are with retraining. For all fusion types (mean, freezing and mixing), we find a significant increase in accuracy after retraining. Mean layer fusion using the WS-2 distance outperforms freezing layers, while random layer mixing performs comparably to averaging. Layer mixing interpolates between neurons a top-k most similar layer pair. Hence, it can change the sign of some of the original incoming weights into the resulting mixed layer. Therefore, it is somewhat surprising that accuracy has remained relatively high, suggesting that similar layers have weights with a shared sign, not only a similar magnitude.

Table 2: **WikiText-2 Test Perplexity Without Fine-Tuning Or Retraining.**

	Trans-XL	GPT-2	GPT	Trans-XL	GPT-2	GPT	Trans-XL	GPT-2	GPT	Trans-XL	GPT-2	GPT
Original	21.28	26.61	67.23	21.28	26.61	67.23	21.28	26.61	67.23	21.28	26.61	67.23
	Layer Pruning via Weight Magnitude			**Global Pruning via Weight Magnitude**			**Randomized SVD**			**Denoising AutoEncoder**		
@ 10%	21.25	25.44	69.33	21.15	25.04	69.54	20.29	25.44	69.33	19.69	23.14	65.14
@ 20%	21.26	27.02	88.19	21.08	27.03	79.33	20.69	27.02	88.19	19.43	24.46	81.08
@ 30%	22.05	35.87	1452.96	21.54	46.15	140.22	21.68	35.87	1452.96	20.57	29.07	921.06
@ 50%	57.12	1627.22	3260.52	53.90	3271.52	2159.42	64.12	1627.22	3145.41	55.07	1258.05	2654.88
@ 70%	3147.31	24946.66	21605.02	901.534	13464.17	18068.86	3679.13	26149.57	22140.12	2958.41	19206.78	19035.38
	Layer Averaging (Euclidean Distance)			**Layer Freezing (Euclidean Distance)**			**Global WS-LF**			**Adjacent WS-LF**		
@ 10%	21.74	25.78	81.14	23.09	28.70	83.44	22.15	25.79	69.29	22.52	25.58	69.90
@ 20%	22.21	29.74	94.80	25.19	30.88	94.32	22.37	27.38	90.70	22.61	27.35	89.77
@ 30%	25.27	38.90	1903.14	27.81	40.01	97.11	24.79	38.18	1533.24	22.82	36.11	1493.37
@ 50%	62.04	1807.31	3724.47	64.38	1944.51	3790.12	61.68	1690.31	3123.39	59.70	1691.23	3357.02
@ 70%	3695.01	2631.52	29117.82	3583.16	23583.10	30258.78	3201.97	25130.30	22448.15	3198.16	25270.21	21732.58

5.2. Language Modelling

We begin by showing how all compression methods suffer in performance when no retraining is used in Table 2. At high compression rates, all compression methods, including out LF techniques (i.e Layer Averaging, Freezing, Gloval WS-LF and Adjacent WS-LF), cannot recover from performance degradation after a one-shot compression step.

In Figure 2, we find an exponential trend in perplexity (note the log-scale y-axis) increase with respect to the compression ratio for layer pruning and global pruning. Interestingly, Transformer-XL can maintain similar performance up to 50% pruning from the pretrained model without any retraining. In contrast, we see that the original OpenAI-GPT is more sensitive and begins to show an exponential increase at 30%. This insight is important for choosing the intervals between compress steps during iterative pruning, likewise for LF and tensor decomposition. Concretely, we would expect that the larger the increase in perplexity between compression steps, the more retraining epochs are needed. We also posit that this monotonically increasing trend in compression is related to the double descent phenomena (Belkin et al., 2019), whereby when more data is added or the model complexity is reduced, the network can fall back into the critical regime region (Nakkiran et al., 2019) and even further into the underparameterized regime. This is reinforced by the fact that a large network such as Transformer-XL contains a smaller global weight norm of fully-connected layers in comparison to GPT and is able to maintain similar performance up to 50% without retraining. Therefore, instead of choosing a uniform amount of compression at each compression step N_c, we allocate more compression earlier but more retraining steps later.

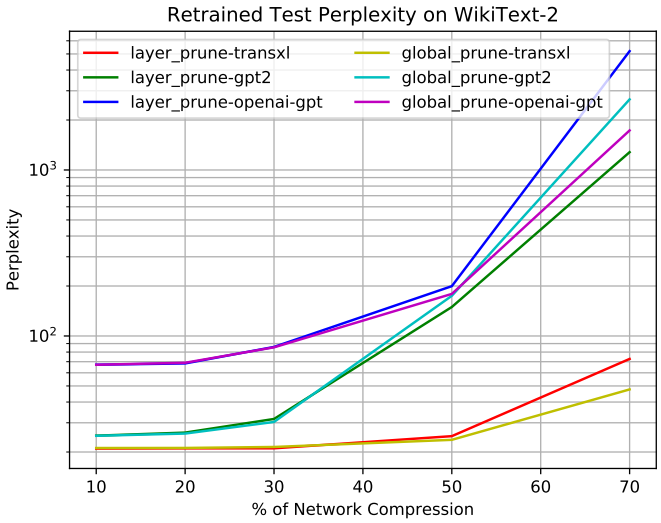


Figure 2: **Wikitext-2: Pruning Without Retraining**

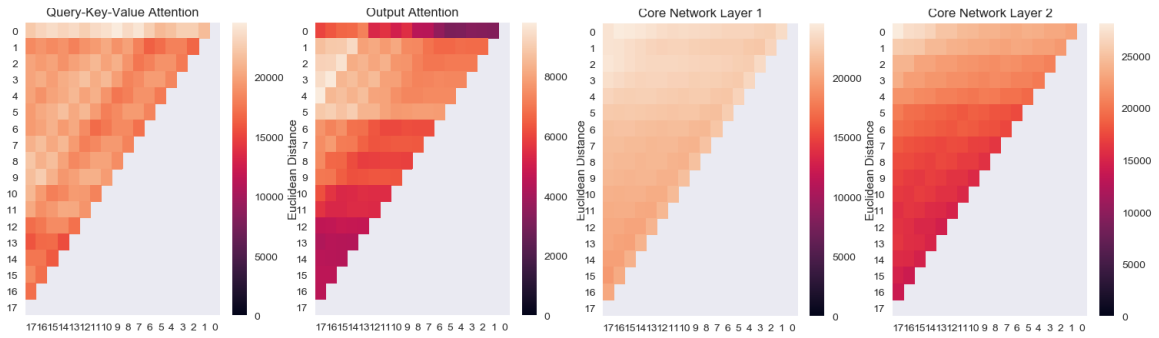
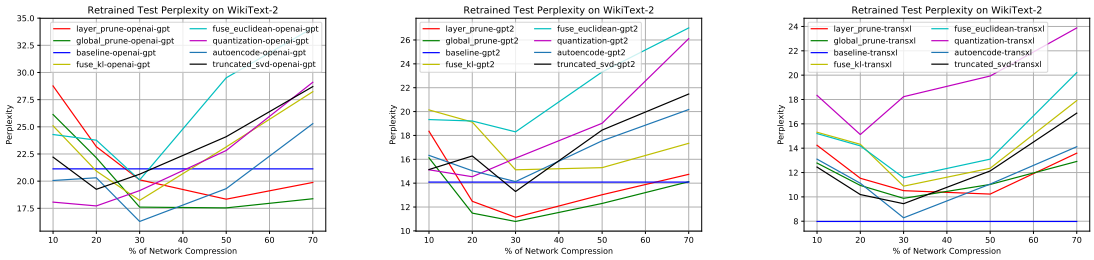


Figure 3: **Euclidean Distance Between Trans-XL Weights**:(1) Query-Key-Value Attention, (2) Output Attention, (3, 4) FC Layer

Figure 3 shows the similarity between pretrained layers on Transformer-XL using the sum of pairwise Euclidean distances. In general, we can see that closer layers have a smaller Euclidean distance. This more pronounced in the output attention (3) and fully connected layers (4) and slightly more sporadic among query-key-value attention weights (1).

Figure 4 shows subfigures of retraining GPT (4(*a*)subfigure), GPT-2 (4(*b*)subfigure) and Transformer-XL (4(*c*)subfigure) with all aforementioned compression methods for GPT, GPT2 and Transformer-XL respectively. Firstly, we find retraining with a sufficient number of compression steps to be worthwhile for drastically reducing the network size while maintaining performance for both structured and unstructured approaches. Past 30% of network reduction wee find a weakly linear increase, in contrast to the exponential increase with no retraining. We find that global pruning generally outperforms layer pruning as it doesn't restrict the percentage of weights pruned to be uniform across layers. This suggests that many layers are heavily pruned while others are preserved. This also coincides with findings from (Zhang et al., 2019) that some layers are critical to maintain performance while removing the remaining layers has little effect on generalization. Table 3 shows the results of LF for compression ratio of 2 using layer averaging (Mean), layer freezing (Freeze) and mixing layers (-Mix) when ranking weight similarity using Euclidean distance (ED), KL and WS distance and CA. For all models CA produces the best results, slightly outperforming WS. Lastly, we compare the best results of Table 3 with *LayerDrop* (Fan et al., 2019, LD), a method that has shown SoTA for structured compression that require no additional overhead, akin to layer fusion. Table 4



(*a*) GPT (*b*) GPT-2 (*c*) Transformer-XL

Figure 4: **Language Modelling Compression on WikiText-2 with Retraining**

shows that the best results obtained from LF outperforms LD when evaluating the models at 50% reduction of their original size. For LD, this corresponds to a dropout rate of 0.5 (best results at 0.5 in Fan et al. (2019)) during retraining on WikiText-2 followed by pruning 50% of layers post-retraining. For all 3 architectures, we find our best LF results improve over LD.

Additional Observations In language modelling, the effects of model reduction typically follow an exponential increase in perplexity for a compression ratio greater than 2 (corresponding to @50%) when no retraining steps are used. Unlike CIFAR10 image classification, language modelling is a structured prediction task that has a relatively large output dimensionality which we posit has an important role in the amount of compression that can be achieved. Yang et al. (2017) have noted the *softmax bottleck* whereby the restriction on the size of the decoder results in information loss when calibrating the conditional distribution, while (Belkin et al., 2019) have also noted the double descent phenomena is dependent on the number of classes. We conjecture that pruning and other such methods can exacerbate this bottlenecking and therefore the compression ratio will be generally lower compared to classification problems with relatively less classes, such as CIFAR-10.

Table 3: **WikiText-2: Perplexity of LF-Retraining @50% reduction**

	Mean	Freeze	Mix
TransXL-KL	12.23	15.02	13.75
TransXL-ED	13.08	17.13	14.88
TransXL-WS	11.48	14.40	12.17
TransXL-CA	**11.13**	13.97	14.73
GPT2-KL	15.56	19.04	15.87
GPT2-ED	16.03	21.14	16.73
GPT2-WS	**13.71**	18.31	13.58
GPT2-CA	14.03	21.28	14.59
GPT-KL	23.57	28.01	24.82
GPT-ED	25.07	29.68	24.73
GPT-WS	19.10	23.17	18.90
GPT-CA	**18.48**	22.01	20.39

Table 4: **WikiText-2: Perplexity With *LayerDrop* & *Layer Fusion***

	TransXL	**GPT2**	**GPT**
Layer Fusion	11.13	13.71	13.58
LayerDrop	14.30	18.93	21.01

6. Conclusion

In this paper we proposed layer fusion for structured model compression. We find that merging the most similar layers during retraining of deep networks leads to competitive performance when compared against the original network, while maintaining layer density. Layer fusion is also competitive with pruning, layer decomposition and knowledge distillation without the use of any additional parameters. Mixing weight matrices during layer fusion performs comparably to layer averaging. We also compared how much compression can be achieved with and without retraining for both tasks and the importance of the number of epochs and compression steps. By using an exponential curriculum schedule to allocate the percentage of compression at each compression step, we find improvements over distributing the compression percentage uniformly during retraining.

Lastly, a clear inflection point was observed in both tasks where performance rapidly decreases for all compression methods and models.

Acknowledgments

This work is based in part on research sponsored by Air Force Research Laboratory (AFRL) under agreement number FA8750-19-1-1000. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright notation therein. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Laboratory, DARPA or the U.S. Government.

References

Eirikur Agustsson, Fabian Mentzer, Michael Tschannen, Lukas Cavigelli, Radu Timofte, Luca Benini, and Luc V Gool. Soft-to-hard vector quantization for end-to-end learning compressible representations. In *Advances in Neural Information Processing Systems*, pages 1141–1151, 2017.

Arturo Argueta and David Chiang. Accelerating sparse matrix operations in neural networks on graphics processing units. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6215–6224, 2019.

Anubhav Ashok, Nicholas Rhinehart, Fares Beainy, and Kris M Kitani. N2n learning: Network to network compression via policy gradient reinforcement learning. *arXiv preprint arXiv:1709.06030*, 2017.

Jimmy Lei Ba, Jamie Ryan Kiros, and Geoffrey E Hinton. Layer normalization. *arXiv preprint arXiv:1607.06450*, 2016.

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

Zihang Dai, Zhilin Yang, Yiming Yang, William W Cohen, Jaime Carbonell, Quoc V Le, and Ruslan Salakhutdinov. Transformer-xl: Attentive language models beyond a fixed-length context. *arXiv preprint arXiv:1901.02860*, 2019.

Mostafa Dehghani, Stephan Gouws, Oriol Vinyals, Jakob Uszkoreit, and Łukasz Kaiser. Universal transformers. *arXiv preprint arXiv:1807.03819*, 2018.

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*, 2018.

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

Peter J Forrester and Mario Kieburg. Relating the bures measure to the cauchy two-matrix model. *Communications in Mathematical Physics*, 342(1):151–187, 2016.

Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. *arXiv preprint arXiv:1803.03635*, 2018.

Nathan Halko, Per-Gunnar Martinsson, and Joel A Tropp. Finding structure with randomness: Probabilistic algorithms for constructing approximate matrix decompositions. *SIAM review*, 53(2): 217–288, 2011.

Song Han, Huizi Mao, and William J Dally. Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv preprint arXiv:1510.00149*, 2015a.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in neural information processing systems*, pages 1135–1143, 2015b.

Babak Hassibi and David G Stork. Second order derivatives for network pruning: Optimal brain surgeon. In *Advances in neural information processing systems*, pages 164–171, 1993.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

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

Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8): 1735–1780, 1997.

Gao Huang, Zhuang Liu, Laurens Van Der Maaten, and Kilian Q Weinberger. Densely connected convolutional networks. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 4700–4708, 2017.

Forrest Iandola, Matt Moskewicz, Sergey Karayev, Ross Girshick, Trevor Darrell, and Kurt Keutzer. Densenet: Implementing efficient convnet descriptor pyramids. *arXiv preprint arXiv:1404.1869*, 2014.

Sergey Ioffe and Christian Szegedy. Batch normalization: Accelerating deep network training by reducing internal covariate shift. *arXiv preprint arXiv:1502.03167*, 2015.

Ehud D Karnin. A simple procedure for pruning back-propagation trained neural networks. *IEEE transactions on neural networks*, 1(2):239–242, 1990.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited. *arXiv preprint arXiv:1905.00414*, 2019.

Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In *Advances in neural information processing systems*, pages 1097–1105, 2012.

Yann LeCun, Yoshua Bengio, et al. Convolutional networks for images, speech, and time series. *The handbook of brain theory and neural networks*, 3361(10):1995, 1995.

Erich Leo Lehmann. *Elements of large-sample theory*. Springer Science & Business Media, 2004.

Yixuan Li, Jason Yosinski, Jeff Clune, Hod Lipson, and John E Hopcroft. Convergent learning: Do different neural networks learn the same representations? In *Iclr*, 2016.

Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3431–3440, 2015.

Stephen Merity, Caiming Xiong, James Bradbury, and Richard Socher. Pointer sentinel mixture models. *arXiv preprint arXiv:1609.07843*, 2016.

Asit Mishra and Debbie Marr. Apprentice: Using knowledge distillation techniques to improve low-precision network accuracy. *arXiv preprint arXiv:1711.05852*, 2017.

Maher Moakher and Philipp G Batchelor. Symmetric positive-definite matrices: From geometry to applications and visualization. In *Visualization and Processing of Tensor Fields*, pages 285–298. Springer, 2006.

Preetum Nakkiran, Gal Kaplun, Yamini Bansal, Tristan Yang, Boaz Barak, and Ilya Sutskever. Deep double descent: Where bigger models and more data hurt. *arXiv preprint arXiv:1912.02292*, 2019.

A Emin Orhan and Xaq Pitkow. Skip connections eliminate singularities. *arXiv preprint arXiv:1701.09175*, 2017.

Antonio Polino, Razvan Pascanu, and Dan Alistarh. Model compression via distillation and quantization. *arXiv preprint arXiv:1802.05668*, 2018.

Alec Radford, Karthik Narasimhan, Tim Salimans, and Ilya Sutskever. Improving language understanding by generative pre-training. *URL https://s3-us-west-2. amazonaws. com/openai-assets/researchcovers/languageunsupervised/language understanding paper. pdf*, 2018.

Jason Sanders and Edward Kandrot. *CUDA by example: an introduction to general-purpose GPU programming, portable documents*. Addison-Wesley Professional, 2010.

Anish Shah, Eashan Kadam, Hena Shah, Sameer Shinde, and Sandip Shingade. Deep residual networks with exponential linear unit. *arXiv preprint arXiv:1604.04112*, 2016.

Sidak Pal Singh and Martin Jaggi. Model fusion via optimal transport. *arXiv preprint arXiv:1910.05653*, 2019.

Emma Strubell, Ananya Ganesh, and Andrew McCallum. Energy and policy considerations for deep learning in nlp. *arXiv preprint arXiv:1906.02243*, 2019.

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

Andreas Veit, Michael J Wilber, and Serge Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pages 550–558, 2016.

Jiaxiang Wu, Cong Leng, Yuhang Wang, Qinghao Hu, and Jian Cheng. Quantized convolutional neural networks for mobile devices. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 4820–4828, 2016.

Zhilin Yang, Zihang Dai, Ruslan Salakhutdinov, and William W Cohen. Breaking the softmax bottleneck: A high-rank rnn language model. *arXiv preprint arXiv:1711.03953*, 2017.

Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov, and Quoc V Le. Xlnet: Generalized autoregressive pretraining for language understanding. *arXiv preprint arXiv:1906.08237*, 2019.

Sergey Zagoruyko and Nikos Komodakis. Wide residual networks. In *BMVC*, 2016.

Chiyuan Zhang, Samy Bengio, and Yoram Singer. Are all layers created equal? *arXiv preprint arXiv:1902.01996*, 2019.

Appendix A. Further Details

Machine Specification. We use an ASUS ROG Strix GL503VS SCAR gaming laptop with an NVIDIA GTX 1070 8GB Graphics Processing Unit (GPU) card for training all models.

Fusing Layers of Unequal Size For a pair of vectorized tensors of column size d and $d+k$, we remove k weights that have the smallest magnitudes from the 2^{nd} tensor until both match. We also considered using PCA and SVD to fix the dimension for each layer pair. However, this is less of an issue in the case of convolutional layers since filters that are most similar tend to be in the same layer. For Transformer models, the same parts of each attention block are fused e.g the key weight tensor from one layer could not be fused with a value weight tensor from another, only another key weight tensor, and these are the same dimensions, hence the same length when vectorized.

Approximating the Covariance Matrix For our experiments, some of the layers can be relatively large. For example, the large GPT-2 a weight matrix from a given hidden layer is $\mathbf{w} \in \mathbb{R}^{2048\times2048}$ and a total of 4,194,304 parameters. We split $\mathbf{W}$ into block matrices to perform covariance estimation. The row d_r and column $d+c$ dimensionality are restricted to $d_r \leq d_r \leq 128$. A submatrix $\mathbf{w}_{i,j}$ can be represented as,

$$\mathbf{w}_{i,j} = \begin{bmatrix} \mathbf{w}_{1,1} & \dots & \mathbf{w}_{1,d_c} \\ \vdots & \ddots & \\ \mathbf{w}_{d_r,1} & & \mathbf{w}_{d_r,d_c} \end{bmatrix}$$

and all submatrices of $\mathbf{W} \in \mathbb{R}^{n\times m}$ are then formed where $m = \text{round}(d_r/128)$ and $n = \text{round}(d_c/128)$. Given a pair of submatrices from two adjacent layers $\mathbf{w}_{i,j}^{n_\ell} \subset \mathbf{W}^{n_\ell}$ and $\mathbf{w}_{i,j}^{n_{\ell+1}} \subset \mathbf{W}^{n_{\ell+1}}$ from layers n_ℓ and $n_{\ell+1}$ respectively, we estimate the covariance similarity between them. This is computed for all adjacent m, n submatrix pairs, assuming that $d_r^{n_\ell} = d_r^{n_{\ell+1}}$ and $d_c^{n_\ell} = d_c^{n_{\ell+1}}$. The complete covariance similarity is then estimated by its mean as,

$$\mathbb{E}[\Sigma_{\mathbf{W}}] = \frac{1}{mn}\sum_{i=1}^{m}\sum_{j=1}^{n} d_{cov}\big(\Sigma_{\mathbf{w}_{i,j}^{n_\ell}}, \Sigma_{\mathbf{w}_{i,j}^{n_{\ell+1}}}\big) \tag{7}$$

The covariance estimation techniques are then applied to compute covariance matrix similarity.