Using Winning Lottery Tickets in Transfer Learning for Convolutional Neural Networks

Ryan Van Soelen
Department of Computer Science
Johns Hopkins University
Baltimore, MD, USA
rvansoe2@jhu.edu

John W. Sheppard
Gianforte School of Computing
Montana State University
Bozeman, MT, USA
john.sheppard@montana.edu

***Abstract*—Neural network pruning can be an effective method for creating more efficient networks without incurring a significant penalty in accuracy. It has been shown that the topology induced by pruning after training can be used to re-train a network from scratch on the same data set, with comparable or better performance. In the context of convolutional neural networks, we build on this work to show that not only can networks be pruned to 10% of their original parameters, but that these sparse networks can also be re-trained on similar data sets with only a slight reduction in accuracy. We use the Lottery Ticket Hypothesis as the basis for our pruning method and discuss how this method can be an alternative to transfer learning, with positive initial results. This paper lays the groundwork for a transfer learning method that reduces the original network to its essential connections and does not require freezing entire layers.**

I. Introduction

When applied to complex problems such as image recognition, neural network architectures tend to be very large, requiring large amounts of computational resources for training, inference, and storage. The large number of parameters also require a sufficiently large data set to properly train the network. This data and hardware burden constrains the applicability of methods based on deep learning. To remedy this issue, many researchers have turned to transfer learning, in which the learned features of a pre-trained network are applied to a new task. A common approach to transfer learning involves using frozen versions of the weights of the lower layers, while retraining the higher layers with the new data. At most, the lower layers are only adjusted through a process of fine-tuning. This work considers an alternative approach in which a large network is distilled to a smaller size, such that it can be retrained from scratch on a new but related problem using the original networks initial parameters.

The basis of this approach stems from the *Lottery Ticket Hypothesis*, introduced by Frankle and Carbin [2]. The hypothesis argues that the strength of a neural network stems only from a subset of their connections. This sub-network, called the *winning ticket*, was by chance initialized in just the right way to allow for good convergence on the data. The authors demonstrate a way of pruning networks into these sparse winning tickets, which can be retrained to achieve a higher performance than the original network.

We propose adapting this lottery ticket-based approach to transfer learning. Rather than transferring the lower-level features of a network, the winning ticket sub-network is extracted and is then retrained on the new data. This allows the important connections to be transferred from the original network, but also allows all layers to be adjusted to the new data set. We observe than the lottery ticket can be pruned to up to 10% of its initial parameters, and that retraining with these small networks can achieve comparable performance to the original architecture when trained on the new data set, depending on the severity of the pruning.

Beyond improved performance of the trained network, there are many other benefits to using this ticket-transfer approach. Assuming the appropriate hardware and software implementations are used, pruned networks are more efficient both in storage and in inference time. This makes the networks more applicable to hardware-limited devices such as mobile platforms or embedded systems.

II. Related Work

A. The Lottery Ticket Hypothesis

As previously stated, the foundation of this work is based on the *Lottery Ticket Hypothesis* [2]. Frankle and Carbin show that randomly initialized dense neural networks can contain sub-networks called *winning tickets*, which when trained in isolation achieve comparable or better test accuracy in a comparable number of iterations. When trained on image data, winning tickets were found in fully-connected networks, convolutional networks, Visual Geometry Group (VGG)-style networks, and Residual Neural networks (ResNets). However, in the case of VGG-style networks and ResNets, the discovery of a winning ticket was conditioned on the training hyperparameters, meaning that some networks may not have winning tickets.

The standard approach for finding the winning ticket is as follows [2]:

1) Randomly initialize a neural network with parameters $\vec{\theta}_0$
2) Train the network for k iterations, resulting in parameters $\vec{\theta}_k$
3) Prune $s\%$ of the network by masking the lowest magnitude parameters to 0

4) Reset the remaining parameters to $\vec{\theta}_0$, creating what is referred to as the *winning ticket*
5) Retrain the winning ticket network using the same data

There are two variants of this process: *one-shot pruning* and *iterative pruning*. In *one-shot pruning*, the network is pruned once to the desired sparsity level. In *iterative pruning*, the network is incrementally pruned and re-trained, until the desired sparsity level is reached. Although taking longer to prune, iterative pruning was found to be capable of creating smaller winning tickets. The existence of a winning ticket implies that the connections outside the ticket do not significantly contribute to the classification performance of the network. Instead, these low-magnitude weights degrade the performance. The results in Frankle and Carbin [2] suggest that, ideally, these connections should be 0, but due to the stochastic nature of the optimization, they instead settle on low-magnitude noise. However, more work needs to be done to confirm this conjecture. Very recently, Frankle and Carbin built upon the original Lottery Ticket Hypothesis, introducing the concept of *late resetting*. For larger networks, the authors found that the winning ticket was more likely to be found and more stable if the weights after a small amount of training were used instead of the initial weights [3]. Late resetting is not explored in this work, but it may be useful to employ when transferring from larger networks.

B. Other Model Compression Techniques

There is a growing interest for designing small, efficient neural networks [4, 5, 12]. Smaller networks take up less space, making them easier to store and deploy to hardware systems. They can also have faster inference times and consume less power. Furthermore, because there are fewer parameters, they may require less data while still avoiding overfitting.

There are two primary approaches for designing small neural networks. The first involves specially designing a new architecture that can achieve levels of performance similar to its larger counterpart. MobileNets and SqueezeNets are examples of this. MobileNet is a specialized image recognition architecture designed for mobile and embedded devices [4]. It is designed to approximate the standard convolutional layer with a more efficient "depth-wise separable convolution." Rather than applying a unique set of filters to each of the M input channels and summing the result, depth-wise separable convolutions apply a single filter to each channel M times and then combine the results using a 1×1 convolution. This has the effect of requiring far fewer arithmetic operations without sacrificing accuracy. Furthermore, two hyperparamters can be adjusted to select the appropriate trade-off between accuracy and efficiency.

Similarly, SqueezeNet uses alternating convolutional layers with 1×1 filters (called squeeze layers) and layers with a mix of 1×1 and 3×3 filters (called expand layers) [5]. This introduces a series of bottlenecks in the network that cut down on the number of parameters. The SqueezeNet architecture was able to meet AlexNet-Level accuracy with $50\times$ less parameters.

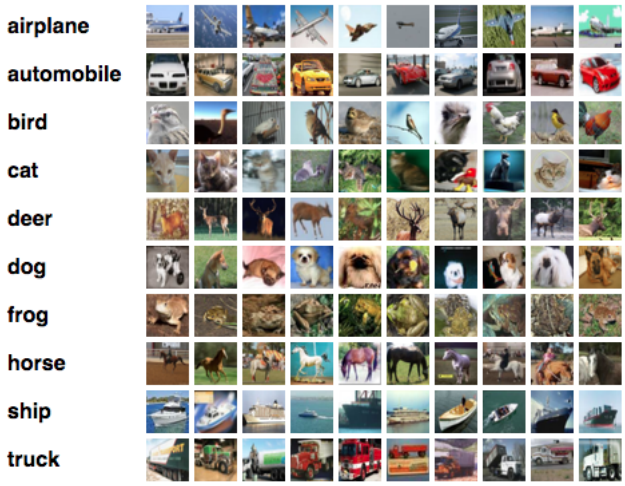


Fig. 1: Ten randomly chosen images from the ten classes in CIFAR-10 [6].

The second approach for creating efficient networks is to incur some sparsity in the network, such that although the network might take up the same *volume* overall, many of its connections are zero and can be ignored. This approach is closer to the work done in this paper. One common way of inducing sparsity in a model is by incorporating regularization into the training process. A common regularizer to promote sparsity is the L1 regularizer [7].

Another approach involves pruning all low magnitude weights after training. It is commonly believed that low magnitude weights are less significant and can therefore be removed without significantly degrading the model [2, 12]. Zhu and Gupta [12] presents a slightly different approach to pruning in which connections are gradually pruned during training rather than after. Periodically, the low magnitude weights of the network are masked to meet a specific sparsity level. That sparsity level is gradually increased over time until the desired sparsity of the network is met. This is similar to the iterative pruning strategy used by Frankle and Carbin [2] to extract winning tickets, but it is done within a single training session rather than over multiple sessions. Zhu and Gupta [12] claim that the networks can be reduced to have 10% of their original number of parameters without a significant loss in accuracy. Furthermore, the sparse networks can consistently out-perform small dense networks with the same memory footprint.

C. Transfer Learning

Besides model compression, our work seeks to investigate how winning tickets trained on one data set can be applied to another. This is very similar to transfer learning, in which some of the parameters of a network (most commonly from the lower layers) pre-trained on source a data set are used in another network for better convergence on a similar target data set. Transfer learning is usually employed when the target data set is too small to train a deep network from scratch, or when computational resources for training are

limited. The transferred parameters are typically frozen while other parameters of the network are trained from random initialization. The transferred parameters can also be fine-tuned to the new data by allowing them to change during training, possibly with a smaller learning rate. The advantage of transfer learning is that general-purpose models can be adapted to new data without having to train the entire architecture from scratch. Furthermore, Yosinski et al. [11] found that initializing a network with transferred parameters from any number of layers can produce a generalization that lasts even after fine-tuning. However, the degree to which transfer learning is applicable depends on how similar the tasks are. Specialization of higher layers on the original task can degrade performance. Furthermore, neurons between layers can become co-adapted, meaning that severing their connection will negatively impact performance if fine-tuning is not done [11].

D. Transfer Learning with Model Compression

Similar to this work, model compression can be performed in conjunction with transfer learning. Molchanov et al. [9] achieves this by iteratively applying pruning as the model is transferred. First, a pre-trained source network is fined tuned on the target data set until convergence. Then, the neurons of the network are pruned based off of a relevancy criteria. The process of fine-tuning and pruning are repeated until the desired trade-off between sparsity and accuracy is met. The novelty of the authors' approach is in their relevancy criteria. Rather than using weight magnitudes, the relevance of each "neuron" is defined as the change in the loss function if the "neuron" were pruned. For this approach, "neuron" can mean an individual weight, a convolutional kernel, or a set of kernels that form a feature map. The authors pruned at the feature map level, pruning one set of feature map kernels each iteration. To efficiently compute this change in loss, the loss function is approximated by its first order Taylor expansion. The authors also normalize the relevancies within each layer to account for different magnitude scales at different layer depths. Using this approach, the authors were able to transfer the VGG-16 network trained on ImageNet to the Caltech-UCSD Birds 200-2011 data set Similar to this work, model compression can be performed in conjunction with transfer learning. Molchanov et al. [9] achieves this by iteratively applying pruning as the model is transferred. First, a pre-trained source network is fined tuned on the target data set until convergence. Then, the neurons of the network are pruned based off of a relevancy criteria. The process of fine-tuning and pruning are repeated until the desired trade-off between sparsity and accuracy is met. The novelty of the authors' approach is in their relevancy criteria. Rather than using weight magnitudes, the relevance of each "neuron" is defined as the change in the loss function if the "neuron" were pruned. For this approach, "neuron" can mean an individual weight, a convolutional kernel, or a set of kernels that form a feature map. The authors pruned at the feature map level, pruning one set of feature map kernels each iteration. To efficiently compute this change in loss, the loss function is approximated by its first order Taylor expansion. The authors also normalize the relevancies within each layer to account for different magnitude scales at different layer depths.

Liu et al. [8] takes a slightly different approach by using a three-stage method for producing a sparse, transferred network. In the first stage, a pre-trained source network is pruned to what is called the *Sparse-SourceNet*. In the second stage, the *Sparse-SourceNet* is modified into a hybrid network called *Hybrid-TransferNet* that splits into two branches. The main branch outputs a prediction the source domain, while the extra branch outputs a prediction in the target domain. Furthermore, the higher layers of the main branch are made dense and randomly reinitialized. As images from the target domain are fed in, the network is trained to minimize the loss of its prediction in the target domain compared to ground truth labels and the loss of its prediction in the source domain compared to what the original source network would have predicted if fed the same image. This hybrid training is done so that the lower levels of the network are can retain implicit knowledge from the source domain that is relevant to the target domain. In the third stage, the main branch of the *Hybrid-TransferNet* is transformed into the *Sparse-TargetNet* and again pruned, since the higher levels of the main branch were made dense in the previous step. The extra branch is still retained during pruning so that the total loss as defined in the *Hybird-TransferNet* can be used when fine-tuning in between pruning iterations.

The common theme between Molchanov et al. [9] and Liu et al. [8] is that the process of training the target network begins using the final weights of the source network. In contrast, this work seeks to investigate whether the latent information in the initial weights of the source network's winning ticket would be useful in transfer learning. In this way, the network can be trained from the ground up using a much smaller network, rather than being fine-tuned from the final weights of the source network. In cases when the size of the target data set is small, the reduced number of parameters is also believed to mitigate against overfitting.

III. Experimental Setup

The goal of this paper is to examine whether winning tickets can be transferred to other data sets. Specifically, we hypothesize that winning tickets can provide a basis for transfer learning, whereby we are able to use the winning tickets in a new but related context to yield comparable classification accuracy with substantial decrease in required computational resources (i.e., time and space). To demonstrate this, the ten classes of the CIFAR-10 data set [6] were randomly partitioned into two data sets of five classes each. Example images of each of the classes are shown in Figure 1. A dense convolutional neural network was trained on Data Set 1, the source data set. Then, the winning ticket of the dense network was extracted using one-shot pruning. Rather than retraining on Data Set 1, the sub-network was retrained on Data Set 2, the target data set. During training, each network was periodically evaluated using a test set. The accuracy of the transferred winning ticket was then compared to that of the original architecture trained

TABLE I: Network architecture for winning ticket transfer.

Input Layer 0	28x28, 3 Channels
Convolutional Layer 1	64 5x5 filters, stride 1
ReLU Activation 1	-
Max Pooling 1	3x3 windows, stride 2
Local Response Normalization 1	depth radius 4
Convolutional Layer 2	64 5x5 filters, stride 1
ReLU Activation 2	-
Max Pooling 2	3x3 windows, stride 2
Local Response Normalization 2	depth radius 4
Fully Connected Layer 3	384 output
ReLU Activation 3	-
Fully Connected Layer 4	192 output
ReLU Activation 4	-
Fully Connected Output Layer 5	5 output
Softmax Activation 5	-

TABLE II: Winning Ticket Network Top-1 Test Accuracy

Network	Mean Accuracy	Standard Deviation	T-value	P-value
25% Sparsity Net	0.797	0.0488	0.942	**0.371**
Dense Net	0.796	0.0479		
50% Sparsity Net	0.792	0.0479	1.84	**0.0982**
Dense Net	0.796	0.0449		
75% Sparsity Net	0.783	0.0538	2.27	0.0491
Dense Net	0.794	0.0470		
90% Sparsity Net	0.768	0.0535	4.98	0.000758
Dense Net	0.791	0.0466		

on Data Set 2. This was repeated for 10 unique splits of the CIFAR-10 classes yielding a 10×2 cross-validation type of design. We then applied target sparsity levels of 25%, 50%, 75%, and 90% for the non-bias parameters in each layer.

Although the method for extracting the winning ticket closely followed that in Frankle and Carbin [2], there were some slight differences. Rather than masking $s\%$ of the entire network's parameters as a whole, each layer was individually pruned by $s\%$. This prevented the process from only pruning layers that overall have lower magnitude weights than other layers. Since bias terms were suspected of behaving differently than other connections, the bias terms of each layer were excluded from pruning. However, the observed network accuracy did not significantly vary in other experiments where the bias connections were included. The most significant difference is the fact that the winning ticket was not trained on the same classes as the original network, but on the classes from Data Set 2.

Table I describes the network architecture used in our experiments, which is based on a design by Krizhevsky and Hinton [6]. All convolutions used padding such that the dimensionality of the feature maps remained the same. The networks were trained using traditional stochastic gradient descent with cross-entropy loss, a learning rate of 0.01, and a batch size of 256.

IV. Results and Discussion

A. Transferring the Winning Ticket

Figure 2 shows the networks' learning curves from one of the five different class splits used. More examples of the learning curves can be found in Appendix A. The results confirm the ability of the pruned network to be re-trained on a new data set without sacrificing much accuracy. In each class split, the winning ticket network was able to be retrained and achieve comparable accuracy to the dense network. However, the winning ticket overall did not out-perform the dense network trained on the same data set. In many cases, both the winning ticket and dense network converge to close to the same accuracy level, depending on the degree of pruning. However, sometimes the winning ticket converges at a slower rate.

The accuracy of both networks at the end of training was compared at different levels of sparsity using the paired sample t-test. This tested whether the difference in mean accuracy over the 10 class label splits was statistically significant (p-value below 0.05). Table II summarizes these findings, marking p-values above 0.05 in bold. The difference in accuracy was shown to be statistically significant for sparsity levels of 0.90 and 0.75, but could not be shown to be statistically significant for sparsity levels of 0.25 and 0.50. This indicates that high levels of pruning will slightly degrade accuracy, but low to moderate levels will not have a significant impact.

There are two factors that might explain these results. The first is that for high levels of sparsity, the winning ticket might be over-pruned. Over-pruning would cut out beneficial parts of the network, causing a degradation of performance. This is supported by the fact that many of the instances where the dense network out-performs the winning ticket is when the ticket is pruned to 90% sparsity. For the given problem; more than 10% of the network may still be needed to fully perform the classification task.

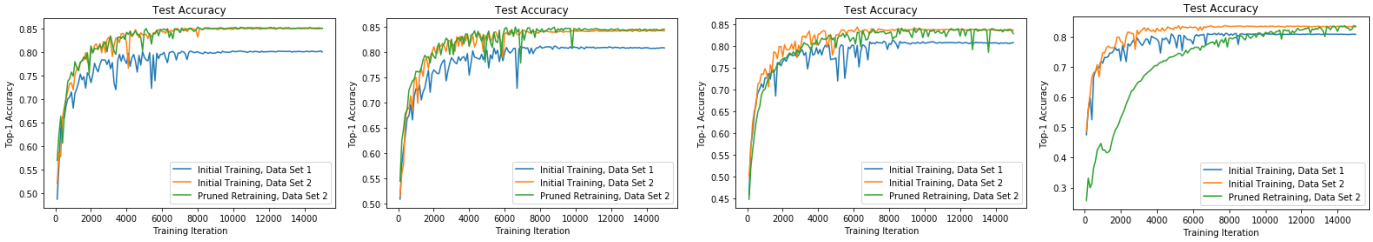
The second factor might be that for this specific problem, a small, randomly initialized network can easily re-learn any latent information that would be transferred by the winning ticket. To evaluate this idea, this experiment would have to be done using either a larger data set like ImageNet [1], or using a deeper, more complex network.

TABLE III: Random Sparse Network Top-1 Test Accuracy

Network	Mean Accuracy	Standard Deviation	T-value	P-value
25% Sparsity Net	0.807	0.0315	1.33	**0.217**
Dense Net	0.811	0.0360		
50% Sparsity Net	0.804	0.0320	5.20	0.000565
Dense Network	0.813	0.0355		
75% Sparsity Net	0.803	0.0318	5.63	0.000324
Dense Net	0.815	0.0362		
90% Sparsity Net	0.799	0.0338	3.55	0.00621
Dense Netw	0.811	0.0346		

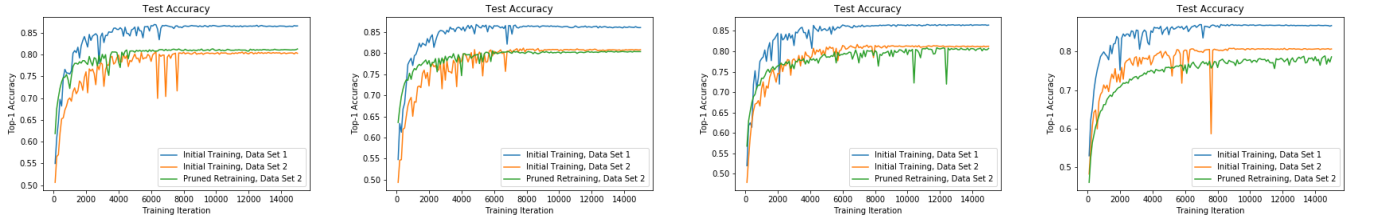
B. Evaluating Randomly Initialized Sparse Networks

According to Frankle and Carbin [2], the performance of the wining ticket depends on whether or not the initial weights from the original training are used when re-training the winning ticket. To verify this requirement, we repeated the previous experiment, but we first randomly initialized the



(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 2: Test accuracy as the network trains. **Data Set 1:** *automobile, airplane, frog, cat, bird*. **Data Set 2:** *truck, ship, dog, horse, deer*



(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 3: Test accuracy as the network trains with a randomly initialized pruned network. **Data Set 1:** *horse, automobile, frog, airplane, dog*. **Data Set 2:** *ship, bird, deer, truck, cat*

winning ticket such that it has the same topology as the pruned original network, but with randomized weights. Figure 3 shows a set of learning curves for a particular class split, and Appendix B shows more examples of the learning curves.

Interestingly, whether or not the initial weights were used did not have a strong impact on the performance of the network. The randomly initialized sparse network achieved a comparable accuracy to the dense network trained on the same data. There are a number of reasons why this might be the case. Since each network is trained on five classes, it might be that the features required to classify those classes are not general enough to be useful when they are transferred. It also might be that for the given problem and architecture, the sparse network was able to learn the necessary features from scratch, regardless of the initialization. Since winning tickets are not always discovered in networks, it is also possible that a strong winning ticket did not exist but the sparse network was still able to sufficiently learn the task.

The paired sample t-test was also performed for this experiment as well, summarized in Table III. It was found that the differences in accuracy between the randomly initialized sparse network and the dense network were statistically significant for sparsity levels greater than or equal to 50%. Interestingly, the p-value dropped sharply between 25% and 50% sparsity. Although both variants of the sparse network tended to perform worse than the dense network at high levels of sparsity, the randomly initialized network's performance was much more sensitive to the sparsity level. This indicates that the transferred initial weights did in fact provide an advantage over random initialization. This effect may have been more pronounced on a more challenging learning task in which the network could have benefited more from the transferred information.

C. Limitations and Future Work

The idea of lottery ticket networks is still very new, and this work is predominately exploratory. Therefore, it is important to point out areas that this study does not address as potential avenues for new research. First, this work only focused on image data, and specifically on transferring networks between different partitions of the CIFAR-10 data set. Work has yet to be done transferring between two distinct image data sets, between non-image data sets, or on data sets with a large number of classes such as ImageNet. Furthermore, only a specific architecture has been explored so far. More work could be done using fully-connected (non-convolutional) networks, networks with skip connections, and recurrent networks. Excluding the work using late resetting, Frankle and Carbin [2] has stated that the existence of winning tickets in ResNets and VGG-style networks is dependent on the choice in training hyperparameters, which suggests that the transferability of winning tickets may also depend on the architecture and hyperparameter choice.

Another simplification made during this work was in setting the number of classes equal for both partitions of the CIFAR-10 data set. This was done to avoid the need to retrain a new output layer with a different shape than the original output layer. In practice, most data sets will have a different number of classes than the original data set. Although the information from the hidden layers would be retained, it is not known

if the randomly initialized output layer would impact the retraining of the rest of the winning ticket. If so, the new output layer might have to be trained using traditional transfer learning before the full winning ticket can be extracted. More work should be done analyzing the theoretical justification for lottery ticket networks and how the winning tickets encode the important information from the data set they were trained on.

To evaluate the practical efficacy of lottery ticket transfer learning, the method should be benchmarked against similar transfer learning techniques that also employ model compression. Comparing this method to similar works also presents the opportunity to consider how some of the ideas from those works can be combined with lottery ticket transfer learning. For, example rather than using the magnitude of each connection to determine its importance, the approximated change in loss could be used similar to Molchanov et al. [9]. Another idea is that a variant of the *Hybrid-TransferNet* introduced in Liu et al. [8] could be used when training the lottery ticket on the target domain. Since the secondary output of the *Hybird-TransferNet* is meant to retain implicit knowledge from the source domain, it might encourage weights to take a path in weight space similar to the path taken when originally trained in the source domain. Finally, recent improvements to the Lottery Ticket Hypothesis like late resetting [3] can also be employed to better enable lottery tickets to be transferred from larger networks.

V. Conclusion

The results of our experiments support our hypothesis that winning tickets can be transferred and thereby speed up training on new (but related) classification problems. While these results are preliminary in nature, we are encouraged by these results.

There are many advantages of transferring winning tickets to new data sets. The resulting networks would take up less memory and have a faster execution time, allowing them to be trained quickly. Since the winning ticket would be sampled from a network pre-trained on a similar task, the ticket's initial conditions might help it train better than random initialization. However, due to the scope of this study, we are only able to advocate tentatively for the merits of this approach. More work needs to be done in a few areas to flesh out this novel concept. Larger, more diverse data sets need to be used.

This method should also be tested on more network architectures as well, including well known high performance networks like Inception [10]. Lastly, more controls should be set to ensure that the pruned network is actually benefiting from the prior training, and is not just learning from scratch. This paper demonstrates that these more involved experiments are worth exploring. If the method does prove to be practical, it will serve as an attractive alternative to standard transfer learning.

References

[1] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei. ImageNet: A Large-Scale Hierarchical Image Database. In *CVPR*, 2009.

[2] Jonathan Frankle and Michael Carbin. The lottery ticket hypothesis: Finding sparse, trainable neural networks. In *International Conference on Learning Representations*, 2019. URL `https://openreview.net/forum?id=rJl-b3RcF7`.

[3] Jonathan Frankle, Gintare Karolina Dziugaite, Daniel M Roy, and Michael Carbin. The lottery ticket hypothesis at scale. *arXiv preprint arXiv:1903.01611*, 2019.

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

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

[6] Alex Krizhevsky and Geoffrey Hinton. Learning multiple layers of features from tiny images. Master's thesis, University of Toronto, 2009.

[7] Baoyuan Liu, Min Wang, Hassan Foroosh, Marshall Tappen, and Marianna Pensky. Sparse convolutional neural networks. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 806–814, 2015.

[8] Jiaming Liu, Yali Wang, and Yu Qiao. Sparse deep transfer learning for convolutional neural network. In *Thirty-First AAAI Conference on Artificial Intelligence*, 2017.

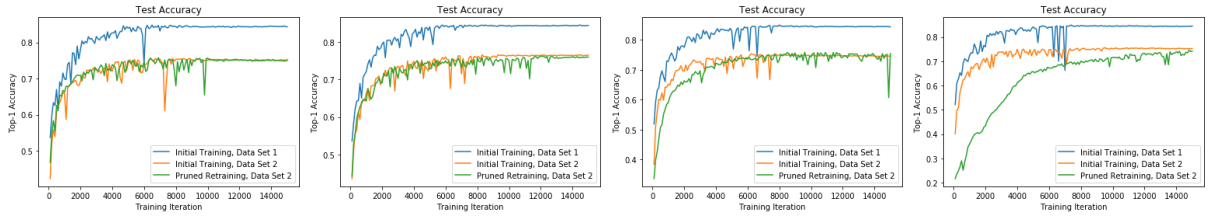
[9] Pavlo Molchanov, Stephen Tyree, Tero Karras, Timo Aila, and Jan Kautz. Pruning convolutional neural networks for resource efficient transfer learning. *arXiv preprint arXiv:1611.06440*, 3, 2016.

[10] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

[11] Jason Yosinski, Jeff Clune, Yoshua Bengio, and Hod Lipson. How transferable are features in deep neural networks? In *Advances in neural information processing systems*, pages 3320–3328, 2014.

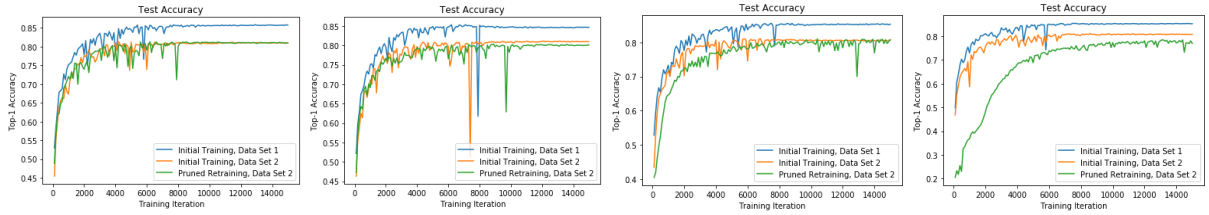
[12] Michael Zhu and Suyog Gupta. To prune, or not to prune: exploring the efficacy of pruning for model compression. *arXiv preprint arXiv:1710.01878*, 2017.

APPENDIX A
TRAINING WITH WINNING TICKETS



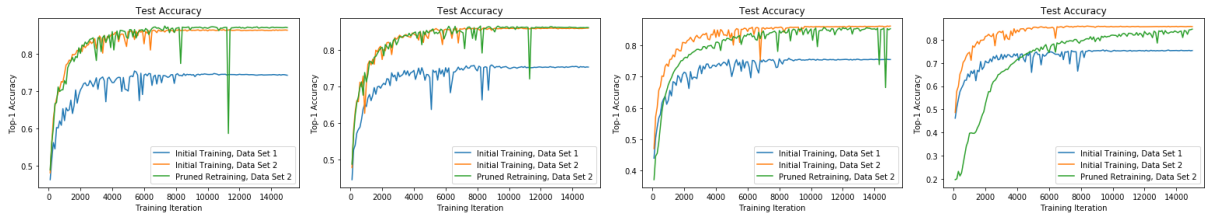
(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 4: Test accuracy as the network trains. **Data Set 1:** *ship, bird, frog, airplane, truck*. **Data Set 2:** *cat, automobile, horse, deer, dog*



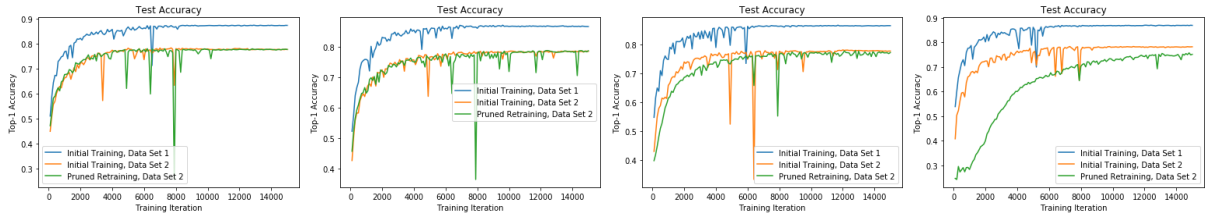
(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 5: Test accuracy as the network trains. **Data Set 1:** *airplane, deer, ship, dog, automobile*. **Data Set 2:** *horse, truck, bird, cat, frog*



(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

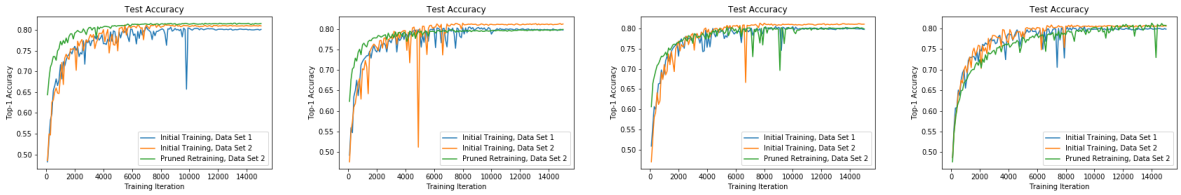
Fig. 6: Test accuracy as the network trains. **Data Set 1:** *frog, dog, airplane, cat, deer*. **Data Set 2:** *truck, ship, bird, automobile, horse*



(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

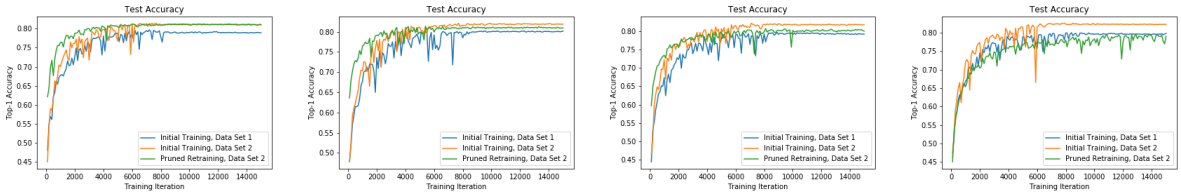
Fig. 7: Test accuracy as the network trains. **Data Set 1:** *frog, dog, ship, airplane, automobile*. **Data Set 2:** *cat, deer, truck, bird, horse*

APPENDIX B
TRAINING WITH RANDOMLY INITIALIZED WINNING TICKETS



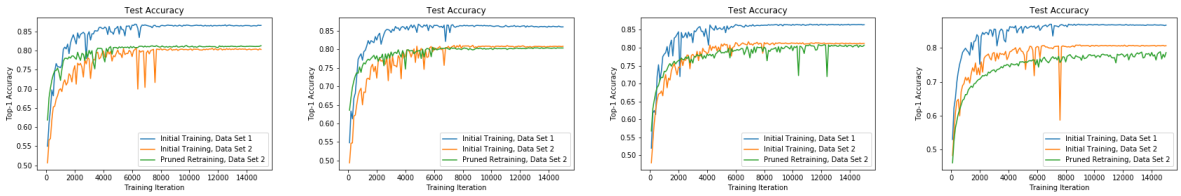
(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 8: Test accuracy as the network trains. **Data Set 1:** *ship, cat, truck, dog, frog*. **Data Set 2:** *airplane, bird, horse, deer, automobile*



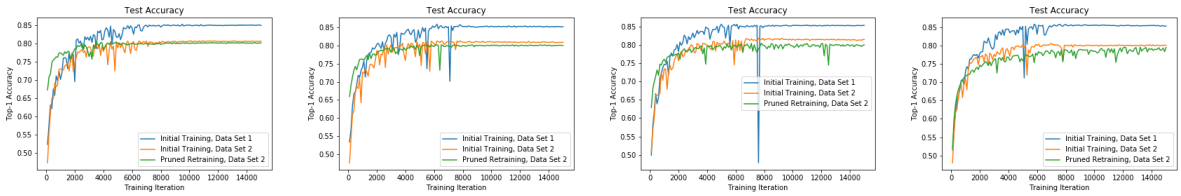
(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 9: Test accuracy as the network trains. **Data Set 1:** *cat, truck, dog, airplane, automobile*. **Data Set 2:** *ship, deer, bird, frog, horse*



(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 10: Test accuracy as the network trains. **Data Set 1:** *horse, automobile, frog, airplane, dog*. **Data Set 2:** *ship, bird, deer, truck, cat*



(a) 25% Sparsity (b) 50% Sparsity (c) 75% Sparsity (d) 90% Sparsity

Fig. 11: Test accuracy as the network trains. **Data Set 1:** *truck, airplane, automobile, dog, deer*. **Data Set 2:** *ship, horse, frog, bird, cat*