System Scaling Challenges of Training a One Trillion Parameter Deep-Learning Model

A.K. Roy
(roy@riftedge.com)

1. Introduction

Deep-Learning (DL) models have grown at roughly 10x per year in terms of model parameters or features over the past few years, as DL practitioners have improved accuracy and solved larger problems. Recent State-of-the-art (SOTA) models include OpenAI's GPT-3 Natural-Language-Processing (NLP) model [1], with 175 billion parameters. DL practitioners will soon push to models with one trillion parameters or more. This blog examines the key system engineering challenges that will need to be addressed, in training such a one-trillion-parameter (OTP) model.

2. Motivations of scaling to a OTP model

The GPT-3 NLP model, even with its SOTA 175 billion parameters, still show a gap of over 10% between model accuracy and humans in many NLP benchmarks. Recent DL scaling-law studies [2] have shown that model accuracy has a power-law behavior with model size, indicating that NLP models need to push beyond one trillion parameters to close the gap. Incidentally, language researchers have commented that the current NLP models, even with their large sizes today, still do not exhibit true real-world language understanding [3], indicating again that NLP models need to continue to add additional features or parameters. Similar gaps between SOTA model-accuracy and the ground-truth exist in other types of models or use-cases, including Visual-Perception or Image-Processing (RCNN) models [4] and Recommendation-Engine (DLRM) models [5]. All the above data indicates that DL practitioners will continue to push model sizes to one trillion parameters and beyond, as they push to deliver better accuracy and real-world understanding, and Artificial-general-intelligence (AGI).

3. Key system challenges of training a OTP model

Training a OTP model with an acceptable Time-to-train (TTT) will need a large Distributed-Training-Cluster or Supercomputer (DLDC), with hundreds of DL Accelerators working together. The key system challenges that need to be addressed for such a DLDC are:

1. Scaling the DL computational capability, in terms of Floating-point operations per second (or FLOPs) to deliver an acceptable TTT
2. Scaling the Accelerator-memory capacity to the necessary amount, to store all the required model state and intermediate values, during the training process
3. Implementing an efficient distributed-training method, to distribute the computing across the DLDC, and deliver high scaling efficiency
4. Scaling the Network bandwidth, to the required level to support the distributed-training method and deliver an acceptable TTT
5. Creating a sufficient large training dataset, to prevent overfitting and degradation of the generalization error of the model
6. Implementing sufficient reliability in the DLDC, to ensure that the training can complete, without encountering severe failures

The following sections will describe each of these system challenges and examine possible solutions.

3.1 Computational capacity needed to train a OTP model

The FLOPs required to training a DL model depends on the type of model and the types of layers in the model.
A simple approach to determining the total FLOPs required for a OTP model is to scale up the FLOPs that were needed of a representative SOTA model, with the commonly occurring layers, using the power-law scaling laws published in [2].

3.1.1. NLP models

For NLP models, a representative SOTA model is the GPT-3 model, with its 175 billion parameters [1]. The training computation takes 6 FLOPs, for each model parameter, for each training token.
Scaling to a OTP model, the total FLOPs required across all epochs of training would be **6.3E+24 FLOPs**. Converting into ExaFLOP-days (1 Exa-FLOP-day = 1E+18*24*3600 FLOPs), **the total FLOPs needed for a NLP OTP model would be 72.9 ExaFLOP-days**.

3.1.2. RCNN models

For RCNN models, a representative SOTA model is the Cascade-RCNN model [4], which has 345 million parameters. The Cascade-RCNN model used 5 TeraFLOPs of training per image across all epochs and used a training dataset of 118K images.
Scaling the dataset to a OTP model, the total FLOPs needed, across all epochs, would be **3.1E+24 FLOPs.** Converting to ExaFLOP-days**, the total FLOPs of computation needed for a RCNN OTP model would be 36 ExaFLOP-days**.

3.1.3. Summary and recommendation

The above calculations show the tremendous computational capability challenge that practitioners face to train a OTP model.
A 1 ExaFLOP/s capable DLDC will take several weeks to complete a training run, assuming 100% scaling efficiency. **A more practicable solution would be a 10 ExaFLOP/s DLDC, which would reduce the TTT to a more tractable two weeks, with a more achievable scaling efficiency of 66%**.
DL Accelerators available in 2021/22 include Nvidia's A-100 GPU, Intel's PointeVecchio GPU and Habana-Gaudi Accelerator, AMD's GPU, GraphCore's Accelerator and Groq's Accelerator [7]. There is also the Cerebras's WSE [7]. The best possible FLOPs performance per Accelerator is about 500 TeraFLOP/s, barring the WSE. **A 10 ExaFLOP/s DLDC would need a minimum of 30K of such Accelerators.**

3.2 Accelerator-Memory capacity needed to train a OTP model

Practitioners today use the Accelerator-Memory to primarily store six type of values during training, to achieve best TTT:

1. Intermediate model values, primarily Activations, with 2 bytes per activation in mixed-precision training
2. Model states which include:
    a. Parameters, with 6 bytes per parameter storing both a 16-bit and a 32-bit version
    b. Parameter gradients used in parameter updates, with 2 bytes per parameter using mixed-precision training
    c. Parameter optimizer states, which are used to modulate the update of the parameters, with 8 bytes per parameter across 2 optimizer values
3. Miscellaneous scratch-pad buffers, such as buffers needed for read-in of input training data and buffers needed for network communication.

This blog will use the NLP model as the representative DL model, to examine the Accelerator-Memory capacity required for a OTP model. Similar analysis can be done for the other model

types, such as the RCNN and DLRM models. With regards to the RCNN models, the input image sizes may be large and the input data will be a key component of the Miscellaneous-buffers space needed. With regard to the DLRM models, the input embedding tables may be large and will be a key portion of the Miscellaneous-buffers space.

3.2.1. Activations

Activations are calculated during the forward-pass phase and need to be either stored or re-calculated for re-use during the backward-pass computation.

The number of activations depends on the model type and its constituting layer types, layer counts, the size of the inputs/training samples and the mini-batch-size.

The GPT-3 model uses 96 MHAT layers, a 2048 input sequence length and a 12288 model-dimension. The GPT-3 model training further used a mini-batch-size of 3.2 million tokens or 1560 sequences. With this mini-batch-size, the training would have stored 75.6 trillion activations.

Scaling to the GPT-3 model to a OTP model, the OTP model would store **2292.2 trillion activations**, and need a memory size of **4585 TB**.

The above analysis shows the tremendous Accelerator-Memory capacity challenge that practitioners face, when training a OTP model, for storing activations.

Accelerators in 2021 will have Accelerator-Memory capacities of about 64GB [7], indicating that practitioners cannot store these values in Accelerator-memory as is, since this would take over 70K Accelerators.

Two alternate methods are Activation-checkpoint and Offload-to-CPU-Memory.

Activation-checkpointing, is a method where only the activations from the compute-intensive layers are stored and activations of the other layers are re-computed on-the-fly when they are needed [16]. Aggressive application of the method can reduce the memory-capacity required by a square-root factor with an increase of about 33% in the FLOPs required. The 4585 TB can thus be to 67.6 TB, with a TTT increase of 33%.

The Offload-to-CPU-Memory offloads the activations to the CPU system-memory, with a representative algorithm being [8]. A minimum Accelerator-to-CPU bandwidth of 25 GB/s is required to sufficiently overlap the read-in of activations the backward-pass computation, such that the Accelerators do not stall.

The analysis in section 3.1 shows that scaling the DLDC to FLOPs required for an acceptable TTT is a tremendous challenge. Any increase in the FLOPs, as required by the activation-checkpoint method, is not palatable. Therefore, **the Offload-to-CPU-Memory method would be recommended approach to storing the activations.**

3.2.2. Model states, including Parameters, their gradients and optimizer states

The parameters and their gradients of an OTP model would need 8TB of space.

Optimizer states of an OTP model would need 8TB of space, for commonly used optimizers such as the ADAM optimizer [9], which store two 32-bit values, namely the momentum and variance of the gradient of the parameter.

3.2.3. Miscellaneous buffers

Miscellaneous buffers are primarily used for the input samples being processed, embedding input tables, network communication, check-pointing and scratch-pad operations. For NLP models, a conservative estimate is to use 2 bytes per model parameter, and so would need 2TB of buffer space.

With regard to DLRM models, the input embedding tables can be very large and run into multi-terabytes, as described in [5]. In such cases, the recommended approach is to store the input tables on the host-CPU system-memory, and assign the input layer to the host-CPU for the

computation, with the rest of the computation pipelined on the Accelerators. The host-CPU FLOPs performance as well as the CPU-to-Accelerator network-bandwidth needs to be scaled up to acceptable levels, such that the Accelerators do not stall, waiting for the activations from the input layer.

3.2.3. Summary and recommendation

In summary, the recommend approach for storing the intermediate value and the model states of a NLP OTP model would be: **(a) offload the activations to the host CPU system-memory (b) store the rest of the data, including parameters, gradients and optimizer states, up to 18TB, in the Accelerator-Memory.**

As described in Section 3.1, a OTP model needs a 10 ExaFLOPs/s DLDC with about 30K 2021 Accelerators to complete training with a tractable TTT of two weeks. If each Accelerator in the DLDC has an Accelerator-Memory of 64GB, then the DLDC would need to be organized and orchestrated into DL Sub-clusters (DLSC) of 600 Accelerators each, to store the 18TBs of data, and the DLDC would have 50 such DLSCs.

3.3 Distributed-Training Method for efficient scaling

Key challenges of the Distributed-Training method are:

1. Full utilization of the DLDC Accelerator resources including the computing, the Accelerator-memory and networking bandwidth, and minimize Accelerator bubbles or idle-time
2. Delivering a scaling efficiency of at least 66%

This blog will examine two methods.

The first method is called the DP-ZERO method. The second method is called the MP-DP method. A representation NLP OTP model is used in the analysis.

3.3.1. DP-ZERO distributed-training method

The DP-ZERO method applies the Data-Parallel (DP) method within the DLSC as well as across the DLSCs. The DP method is a SIMD approach, where the training batch is spliced into mini-batches and each Accelerator trains the entire model on its own mini-batch in parallel with other Accelerators, with an averaging of the parameters at the end of each step or epoch. Further, the 'Zero' algorithm described in [10] is applied on storing and fitting the model states into the available 38.4TB of Accelerator-memory across the 600 Accelerators in the DLSC. With the Zero algorithm, each Accelerator computes the entire model but only stores a portion or a partition of the model. Accelerators read-in model parameters and activations of partitions that they do not own, from the Accelerators owning that the partition data, when needing the data. The DP-ZERO method further applies the DP method across the DLSCs, and each DLSC computes the model on its own mini-batch, in parallel with the other DLSCs.

3.3.2. MP-DP distributed-training method

The MP-DP method applies the Model-Parallel (MP) method within the DLSC and the DP method across the DLSCs. The MP method is a MIMD approach, where the model is split 'vertically' or otherwise, and each slice is assigned to an Accelerator, and the Accelerators train the model in a pipeline fashion. Thus, the model is split and pipelined across the 600 Accelerators in a DLSC. Either the 'Pipedream' algorithm described in [11] or the 'GPipe' algorithm described in [12] can be applied to pipeline the processing, to reduce the startup time as well as minimize Accelerator idle-time. Both methods split the mini-batches into micro-batches and pipeline the micro-batches. The Pipedream algorithm is superior to the GPipe algorithm on the minimization of the Accelerator idle-time, at the cost of increased complexity and Accelerator-memory usage for maintaining versions of parameters. Since TTT is the more significant challenge than Accelerator-memory space in training a OTP model, the Pipedream approach would be the

recommended approach. The MP-DP method further applies the DP method across the DLSCs, with each DLSC computing the model on its own batch, in parallel with the other DLSCs.

3.3.3. Summary and recommendation

DP-ZERO method is superior to the MP-DP method with regard to TTT but needs higher network bandwidth. Since TTT is the most pressing challenge, **the DP-ZERO method would be the recommended distributed-training method**. The MP-DP method is best for infrastructures where the network bandwidth is constrained.

In the use-case where the model has a layer that is too large for the layer matrix-multiplication tensors to fit into the 64GB Accelerator-memory available space on a single Accelerator, even with a batch-size of one, the DLSC needs to be further sub-divided into mini-DLSCs of appropriate number of Accelerators. The SUMMA or 2.5D matrix-multiplication algorithm needs to be used to split the matrices across the Accelerators in the mini-DLSC [13], and the mini-DLSCs would need to implement either the DP or the MP approach.

3.4 Accelerator-to-Accelerator Network bandwidth required

Figure 1 shows a block-diagram of the DLDC with its DLSCs. To analyze the network bandwidth needed, the formula of ***n*β*log(p)*** from [14] is used, to calculate the time taken for a broadcast transfer, where ***n*** is the amount of data broadcast, ***β*** is the network bandwidth and ***p*** is the number of nodes.

The bottleneck for communication within the DLSC, in the DP-ZERO method, is the broadcast of the parameters and activations from the Accelerator owning the data to the other 599 Accelerators in the DLSC, ahead of the backward-pass computation of layer. For a NLP OTP model, a minimum ingress network bandwidth of 208 GB/s with a 50% utilization assumption, or a **bi-directional network bandwidth of 416 GB/s, per Accelerator, would be required for within DLSC traffic.**

The bottleneck for communication across DLSCs, in the DP-ZERO method, is the read-in of parameters into an Accelerator in a DLSC, at the end of a step, from corresponding Accelerators in the 49 other DLSCs, such that the parameters can be averaged. For a NLP OTP model, a minimum ingress network bandwidth of 54GB/s or **a bi-directional bandwidth of 108GB/s, per Accelerator, would be required for across DLSC traffic.**

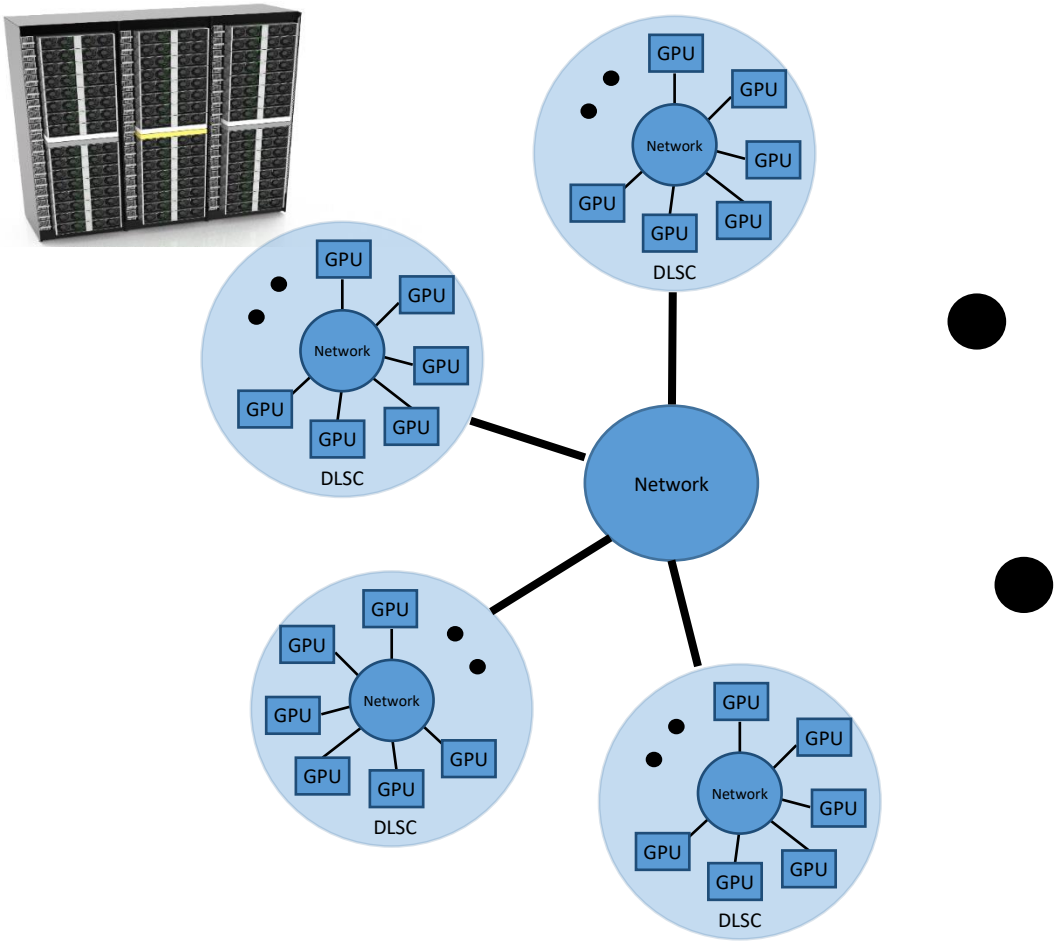

Figure 1. Block-diagram of the DLDC with its DLSCs.

3.5 Creating large-enough training datasets

Two learning approaches are generally used for training, with respect to the training dataset, namely (a) fully supervised learning with a single training dataset with labels (b) transfer learning with a large training dataset with or without labels for 'coarse supervised' training and a smaller training dataset with labels for 'fine-tuning' training. In both approaches, a large enough training dataset must be created to prevent over-fitting and degradation of the generalization error of the model.

Several scaling studies, such as [2, 15], have shown that a power-law relationship exists between generalization error and the training dataset size.

For NLP models, the dataset size is related to the model-size with an exponent of 0.74 [2], and so an 10x increase in model parameters needs a 5.5x increase in the dataset size. The GPT-3 SOTA baseline model used a dataset size of 238 billion tokens for its 175 billion parameters [1]. **A NLP OTP model would need a dataset of 864 billion tokens.**

For Image models, the model-size is related to the training dataset size with an exponent of 0.57[15], and so an 10x increase in training dataset size will support a model size increase of 3.7x. The Cascade-RCNN A SOTA baseline model used a dataset size of 118E+3 images from the COCO dataset, for its 345 million parameters. **A RCNN OTP model would need a training dataset size of about 120 billion images.**

The key challenges for practitioners on creating sufficiently large training dataset are:

a) Methods to create and check the quality of the large dataset. Checks must include incorrect or missing labels, malformed data such as incorrect resolution of images, contamination, duplication with test or evaluation data and unintended biases such as biases against gender, race or country in the data.
b) Ability to store and delivering the large dataset to the DLDC, in a performant and TCO-optimized manner

SOTA methods for creating and cleaning training large datasets for NLP models are described in [1] and [18]. The common NLP training datasets include the Common Crawl dataset [19] and Wikipedia.

SOTA methods for creating and cleaning large training datasets for RCNN models are the methods described in [20].

3.6 Checkpointing and Reliability technology

With the TTT in the order of two weeks or more, efficient checkpointing methods need to be used, to ensure that the training can be restarted on the event of a severe software or hardware failure. **A representative efficient checkpointing method is the 'DeepFreeze' method** described in [17].

Additionally, the DLDC needs to monitor the critical components of the cluster, including the networking components, Accelerators, CPU system-memory and storage SSDs for incipient failures and take appropriate actions to prevent a severe hardware failure. Overall, appropriate checkpointing and reliability methods need to be implemented such that the Mean-time-between-failure (MTBF) of the DLDC far exceeds the TTT.

4. Summary and concluding remarks

Deep-Learning practitioners will face six primary system scaling challenges as they push to models with one trillion parameters and beyond, namely FLOPs required, memory, network bandwidth, training dataset and reliability.

The blog article shows that a 10 ExaFLOP/s Distributed-Training-Cluster (DLDC) will be needed to deliver FLOPs required to achieve a tractable time-to-train of two weeks. To store the model

states and intermediate values, sub-clusters (DLSC) of 600 Accelerators with each Accelerator having a memory capacity of 64GB is required, along with the offloading the activations to the host-CPU system-memory. The ‘DP-ZERO’ distributed-training method is the recommended method to distribute and orchestrate the training as well as the memory-storage with the DLSCs and across the DLDC. High-performance networking is needed, with a minimum network-bandwidth of 410GB/s per Accelerator for within the DLSC traffic and a minimum network bandwidth of 104GB/s per Accelerator for traffic between the DLSCs. A sufficiently large training dataset is required to prevent overfitting and appropriate checkpointing methods need to be used to ensure that the training can complete, without severe errors.

References

[1] Tom B Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, et al. Language models are few-shot learners. arXiv preprint arXiv:2005.14165, 2020.

[2] Jared Kaplan, Sam McCandlish, Tom Henighan, Tom B Brown, Benjamin Chess, Rewon Child, Scott Gray, Alec Radford, Jeffrey Wu, and Dario Amodei. Scaling laws for neural language models. arXiv preprint arXiv:2001.08361, 2020.

[3] Emily M. Bender and Alexander Koller. 2020. Climbing towards NLU: On meaning, form, and understanding in the age of data. In Proc. of ACL.

[4] Cai Z, Vasconcelos N (2019) Cascade R-CNN: high quality object detection and instance segmentation. IEEE Trans Pattern Anal Mach Intell. https://doi.org/10.1109/tpami.2019.2956516

[5] U. Gupta, X. Wang, M. Naumov, C. Wu, B. Reagen, D. Brooks, B. Cottel, K. M. Hazelwood, B. Jia, H. S. Lee, A. Malevich, D. Mudigere, M. Smelyanskiy, L. Xiong, and X. Zhang, "The architectural implications of facebook's dnn-based personalized recommendation," CoRR, vol. 1906.03109, 2019.

[6] Paulius Micikevicius, Sharan Narang, Jonah Alben, Gregory Diamos, Erich Elsen, David Garcia, Boris Ginsburg, Michael Houston, Oleksii Kuchaiev, Ganesh Venkatesh, and Hao Wu. Mixed precision training, 2017.

[7] Linley Gwennap and Mike Demler. 2020. A Guide to Processors for Deep Learning. https://www.linleygroup.com/report_detail.php?num=65

[8] Bharadwaj Pudipeddi, Maral Mesmakhosroshahi, Jinwen Xi, and Sujeeth Bharadwaj. Training large neural networks with constant memory using a new execution algorithm. ArXiv, abs/2002.05645, 2020.

[9] Diederik Kingma and Jimmy Ba. Adam: A method for stochastic optimization. In ICLR, 2015.

[10] Samyam Rajbhandari, Jeff Rasley, Olatunji Ruwase, and Yuxiong He. Zero: Memory optimization towards training a trillion parameter models. arXiv preprint arXiv:1910.02054, 2019.

[11] Aaron Harlap, Deepak Narayanan, Amar Phanishayee, Vivek Seshadri, Nikhil Devanur, Greg Ganger, and Phil Gibbons. Pipedream: Fast and efficient pipeline parallel dnn training. arXiv preprint arXiv:1806.03377, 2018.

[12] Yanping Huang, Youlong Cheng, Ankur Bapna, Orhan Firat, Dehao Chen, Mia Chen, HyoukJoong Lee, Jiquan Ngiam, Quoc V Le, Yonghui Wu, and Zhifeng Chen. Gpipe: Efficient training of giant neural networks using pipeline parallelism. Advances in Neural Information Processing Systems 32, pages 103–112, 2019

[13] R. A. van~de Geijnv and J. Watts. SUMMA: Scalable Universal Matrix Multiplication Algorithm. In LAPACK Working Note 99, technical report, University of Tennessee, 1995.

[14] Rajeev Thakur, Rolf Rabenseifner, and William Gropp. Optimization of collective communication operations in MPICH. The International Journal of High Performance Computing Applications, 19(1):49–66, 2005.

[15] Joel Hestness, Sharan Narang, Newsha Ardalani, Gregory Diamos, Heewoo Jun, Hassan Kianinejad, Md. Mostofa Ali Patwary, Yang Yang, and Yanqi Zhou. Deep learning scaling is predictable, empirically, 2017.

[16] Tianqi Chen, Bing Xu, Chiyuan Zhang, and Carlos Guestrin. Training deep nets with sublinear memory cost. arXiv preprint arXiv:1604.06174, 2016.

[17] Bogdan Nicolae, Jiali Li, Justin Wozniak, George Bosilca, Matthieu Dorier, et al.. DeepFreeze: Towards Scalable Asynchronous Checkpointing of Deep Learning Models. CCGrid'20: 20th IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, Nov 2020, Melbourne, Australia. hal-02543977.

[18]Dmitry Lepikhin, HyoukJoong Lee, Yuanzhong Xu, Dehao Chen, Orhan Firat, Yanping Huang, Maxim Krikun, Noam Shazeer, and Zhifeng Chen. 2020. Gshard: Scaling giant models with conditional computation and automatic sharding. arXiv preprint arXiv:2006.16668.

[19]http://commoncrawl.org/the-data

[20]Mahajan, D., Girshick, R., Ramanathan, V., He, K., Paluri, M., Li, Y., Bharambe, A., and van der Maaten, L. Exploring the limits of weakly supervised pretraining. arXiv preprint arXiv:1805.00932, 2018.