# Reproducibility Report - One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers

**Varun Gohil**[*]
IIT Gandhinagar
gohil.varun@iitgn.ac.in

**S. Deepak Narayanan**[*]
IIT Gandhinagar
deepak.narayanan@iitgn.ac.in

**Atishay Jain**[*]
IIT Gandhinagar
atishay.jain@iitgn.ac.in

## Abstract

The lottery ticket hypothesis states that smaller subnetworks within a larger deep network can be trained in isolation to achieve accuracy similar to that of original network, as long as they are initialized appropriately. However, whether these subnetworks or winning tickets are transferable across datasets and optimizers remains unclear. The paper "One ticket to win them all:generalizing lottery ticket initializations across datasets and optimizers" empirically shows that these winning tickets are transferable. We reproduce the results in the paper from scratch by implementing all the experiments. Our results support the original paper's claim of the winning ticket initializations being transferable. While the paper is replicable, we find that reproducing the paper requires access to large amount of computing resources for generating the winning tickets. Hence we also open-source the winning tickets we find, so others can avoid the compute-intensive procedure of generating them.

## Track : Replication

## 1 Introduction

Prior works have shown that 90% of the parameters of a neural network can be eliminated without compromising accuracy [1; 2]. Eliminating unnecessary parameters by techniques like pruning reduces the computation requirements and energy consumption of neural networks thereby making inference more efficient. The procedure for pruning networks involves training the entire neural network and eliminating the least important weights after the training phase has been completed. However, if the number of parameters in a neural network can be reduced, why not train the pruned network itself and make training phase more efficient?

Pruned networks were not trained from scratch as previous works [2; 1] mention that when pruned networks are trained from scratch they achieve lower accuracy when compared to a network which is pruned after training. However, the recently proposed lottery ticket hypothesis states the following: *"A randomly-initialized, dense neural network contains a subnetwork that is initialized such that—when trained in isolation—it can match the test accuracy of the original network after training for at most the same number of iterations"* [3]. These subnetworks along with the appropriate

[*]Denotes equal contribution.

initializations are referred to as the *winning tickets*. If true, the lottery ticket hypothesis implies that pruned networks can be trained from scratch to achieve accuracy commensurate to the accuracy of original network as long as the pruned network is initialized appropriately.

Unfortunately, finding these winning ticket initializations requires one to iteratively prune the network which is a computationally expensive procedure. One can potentially avoid this procedure if one can reuse the same winning ticket initialization across multiple datasets and optimizers. However, the answer to the question of whether these winning ticket initializations generalize to the spectrum of datasets and optimizers remains obscure. The paper we reproduce, *"One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers"* [4] provides empirical evidence that these winning ticket initializations generalize across multiple datasets as well as optimizers[2].

As a part of the NeurIPS Reproducibility Challenge's *Replication Track*, we replicate the work done by [4] and investigate if the winning ticket initializations are generalizable across datasets and optimizers. The target questions of our work are as follows:

- Do winning ticket initializations generalize within same data distribution?
- Do winning ticket initializations generalize across datasets?
- Do winning ticket initializations generalize across optimizers?

In this report, Section 2 describes the techniques we used for our experiments and their implementation. Section 3 describes the experimental setting and computing resources we use. Section 4 describes the efforts needed to replicate the results in terms of computing resources required, development time and contact with authors. Further, in Section 5 we present and discuss our results. We open source the code we use for our experiments[3]. Finally, as finding these winning tickets is computationally expensive, we open-source the winning tickets we found for usage by the community[4].

## 2 Methodology

### 2.1 Iterative Pruning and implementation

There are two widely used methods for pruning - one shot pruning and iterative pruning. Suppose we want to prune $p\%$ of a network. In one shot pruning, we first train the network, then prune $p\%$ of the weights and finally reset the weights to the original initialization that the network had before training. In iterative pruning the network is trained, pruned and reset every round for $n$ rounds. As can be observed, at the end of each round, $p^{\frac{1}{n}}\%$ of the weights that survived the previous round are pruned. In this work we use iterative pruning for pruning the neural networks. We use iterative pruning as prior work [3] shows that it finds winning tickets that match the accuracy of original network at higher pruning fractions when compared to one shot pruning.

In our implementation of iterative pruning we set the value of the parameters to be pruned to be zero before each forward pass. This automatically ensures two things: (a) the forward pass is on the pruned network and (b) the gradients are computed on the pruned network. Note that the gradients used in updating the weights may make the pruned weight values non-zero. But this is not an issue since we reset the weights again before the next forward pass.

### 2.2 Late Resetting

In the original paper on lottery ticket hypothesis [3], the authors reset the weights after each pruning iteration to the original initialization that the network had before training. They report that learning rate warm-up is necessary to find winning tickets on larger models. However, a recent work [5], reports that re-initializing the weights to the weights after the training iteration $k$, where $k$ is typically much smaller than the total training iterations, performs consistently better in producing winning tickets and also removes the need for learning rate warm-up. We employ late resetting of 1 epoch in all the experiments as used by the authors [4].

[2]Authors used anywhere in this paper refers to the authors of the paper that we reproduce [4]

[3]The code base can be found at `github.com/varungohil/Generalizing-Lottery-Tickets`

[4]The winning tickets can be found in this `Google Drive folder` (hyperlinked)

### 2.3 Global Pruning and Local Pruning

A neural network can be pruned either in a global manner or in a local manner. When pruning in a global manner, the weights of all layers of the network are pooled together and then a fraction of weights are removed from this global pool. In local pruning, the same fraction of weights are removed in each layer for all the layers. In our work we use global pruning as used in the paper we are reproducing [4].

### 2.4 Random Masks

Winning tickets, contain information about two key aspects of the subnetwork: the structure of the sparse neural network as well as the initialization of the parameters. The structure of the subnetwork is stored in form of a mask which is a binary vector that is multiplied with the network's weights, to set the pruned weights to zero. Prior works have preserved the structure of the mask while randomly initializing the weights for random tickets. The authors [4] empirically demonstrate that the structure of the subnetwork contains significant information. Hence for the random ticket the authors apply a random mask to the network and initialize the parameters randomly. For implementing the random ticket baseline, we generate random masks by globally permuting the winning masks as mentioned in [4].

## 3 Experimental Settings

We implement the code base using PyTorch [6]. We use the inbuilt model definitions, optimizers, datasets of PyTorch for our experimentation.

### 3.1 Models

For all our experiments, we use one of the two network architectures: ResNet50 and a modified VGG19.

In the case of the modified VGG19 architecture, we remove all the fully connected layers from the network. Following the last convolutional layer, we add a global-average-pooling layer. Finally, we add a linear classification layer from the global average pool to the number of output classes. We use the ReLU non-linearity and perform batch normalization after each convolutional layer. For our experiments, we initialize all the convolutional layers using Xavier normal initialization and the biases to 0. We set batch norm weights and bias parameters to 1 and 0 respectively. We train all the VGG19 models for 160 epochs and anneal the learning rates by a factor of 10 at the $80^{th}$ and the $120^{th}$ epochs.

We use the standard ResNet50 architecture that was proposed in [7]. We use the Kaiming normal initialization for convolutional layers, which is also the default initialization for ResNets in PyTorch. We train all the ResNet models for 90 epochs. We anneal the learning rates by a factor of 10 at the $50^{th}$, $65^{th}$ and $80^{th}$ epochs.

The initializations, number of epochs, learning rate annealing schedules are in accordance to [4] to maintain consistency of experiments.

### 3.2 Optimizers

We used two optimizers for our experiments - The Adam optimizer and Stochastic Gradient Descent (SGD) optimizer. We use Adam with a learning rate of 0.0003 with betas 0.9 and 0.999 and a weight decay of 0.0001. We use SGD with a learning rate of 0.1, with a momentum of 0.9 and a weight decay of 0.0001. We use hyperparameters provided by authors to maintain consistency with the paper we are reproducing [4].

### 3.3 Datasets

We use 4 datasets for our experiments - CIFAR10 [8], CIFAR100 [8], SVHN [9] and FashionMNIST [10]. These datasets are diverse in terms of grayscale vs. color images, input size, number of output classes, and training set size. For all these datasets for data augmentation we perform random horizontal flips and random crops of size 32 with a padding of 4.

### 3.4 Pruning Settings

We perform iterative pruning for 30 pruning iterations and use a 20% pruning rate. For our experiments we use magnitude-based pruning. The weights with the magnitudes in the lowest 20% of remaining non-zero weights are removed after each iteration. While performing iterative pruning, we use late-resetting of 1 epoch.

### 3.5 Computing Resources

We run our experiments on three GPUs - Nvidia P100, Nvidia K80 and Nvidia GTX 1080. The Nvidia P100 and Nvidia K80 machines had a 16 core Intel processor and 15GB RAM, while the Nvidia GTX 1080 machine had a 32 core Intel processor with 256 GB of RAM.

## 4 Cost of Reproducibility

The authors of the original paper [4] did not release their code. We replicate the results by implementing the all experiments from scratch. We did not experience significant difficulty in developing the code base we use for our experiments. We believe that a person having experience with PyTorch can implement the code without major challenges. Further, we also contacted the authors via email. We inquired about the data-augmentations used while training the networks as they were not mentioned in the original paper. Further we contacted them to understand the concept of random masks.

Replicating the results required a significant amount of computing resources. We experienced that the compute resources provided by Code Ocean were not sufficient for the experimentation and hence we performed the experiments on Google Cloud. Further, finding winning tickets for larger datasets is computationally expensive, with the authors using 16 GPUs [4]. As we did not have access to such a large amount of computing resources, we only replicate the results reported on smaller datasets like Cifar-10, Cifar-100, SVHN and FashionMNIST. We could not conduct experiments for larger datasets like ImageNet (10 million images) [11] and Places365 (8 million images) [12] as we were severely limited by compute capability and the time allotted for the reproducibility challenge. Overall, we used approximately $500 worth of Google Cloud credits for our experimentation.

The process of generating winning tickets is time-consuming as well. Training a ResNet50 model for 90 epochs using Nvidia P100, the fastest GPU we used, takes approximately 33 minutes. Similarly, training the VGG19 model for 160 epochs using Nvidia P100 takes approximately 43 minutes. For our experiments we trained a ResNet50 model 450 times and a VGG19 model 540 times. All the experiments would take approximately 634 hours (26 days) to run sequentially. To complete the experiments in time we scheduled multiple experiments parallelly on Google Cloud.

We open-source our code base for reproducing the results of [4]. Along with our code base, we also open-source the winning tickets we found during our experimentation. We hope this will help the community avoid expensive and time-consuming computation, as these winning tickets can directly be used for inference and can be studied to improve our understanding of lottery tickets.

## 5 Results and Discussion

The original paper reports results for 3 experiments, each concerning a target question we mentioned in Section 1. For each experiment, we plot the test accuracy at convergence as function of fraction of pruned weights. Owing to the compute and time constraints mentioned in Section 4, we could only replicate the results with 1 random seed.

### 5.1 Transfer within same data distribution

With this experiment, we aim to investigate if the winning ticket initializations generalize within the same data distribution. For this experiment, we divide the CIFAR-10 dataset into 2 halves - CIFAR-10a and CIFAR-10b. Both these halves contain 25,000 training images, having 2500 images of each class. We find the winning ticket initialization for CIFAR-10a using SGD and verify if it generalizes to CIFAR-10b. As our baselines, we use the CIFAR-10b winning ticket initialization with SGD and random tickets. We perform this experiment for both, VGG19 and ResNet50 architectures,

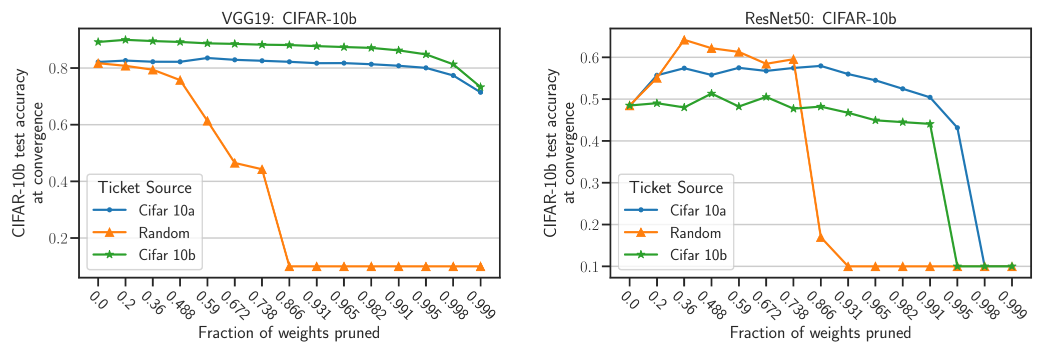

Figure 1: Transfer of winning ticket initializations within same data distribution

Our results are presented in Figure 1. The results show that winning tickets found on CIFAR-10a generalize well to CIFAR-10b. We also see that while using ResNet50, for low pruning fractions random ticket provides better accuracy than winning tickets found using CIFAR10-a and CIFAR10-b. The same phenomena is reported in the original paper [4]. Our results support the hypothesis presented in original paper that ResNet50 winning tickets are sensitive to smaller datasets at low pruning fractions.

## 5.2 Transfer across optimizers

With this experiment, we aim to investigate if the winning ticket initializations generalize across optimizers. For our experiments, we use the modified VGG19 architecture discussed in Section 3. We find the winning tickets for CIFAR-10 dataset using both Adam and SGD optimizers and analyze the effect on accuracy when the ticket generated using one optimizer is further trained using another optimizer. Our results show that even after transfering tickets from SGD to Adam and vice-versa the accuracy of the tickets was comparable to when the tickets were trained using the same optimizer without any transfer. This supports the claim made in the original paper that VGG19 winning tickets are *optimizer-independent*.

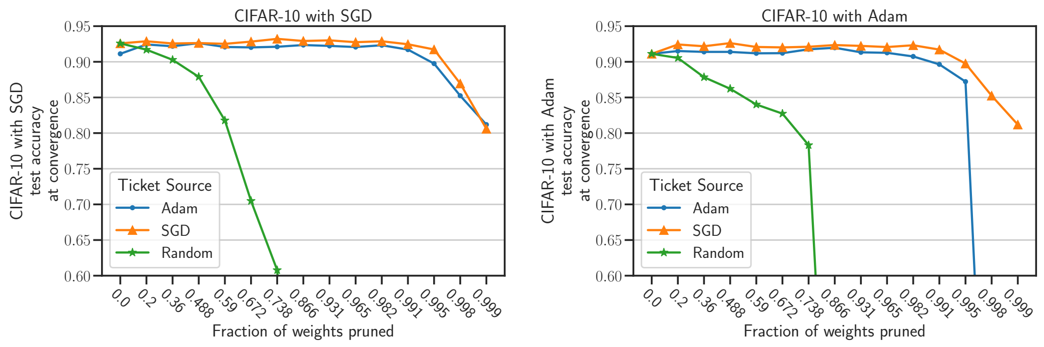

Figure 2: Transfer of winning ticket initializations across optimizers

## 5.3 Transfer across datasets

With this experiment, we aim to investigate if the winning ticket initializations generalize across datasets. For our experiments, we used ResNet50 and the modified VGG19 architectures. We train the models on Cifar-10, Cifar-100, FashionMNIST and SVHN datasets with SGD optimizer.

Our results also reveal the key trends which the authors discuss in the original paper. Firstly, we see in Figures 3 and 4 that individual winning tickets show accuracy similar to that of winning ticket generated on the target dataset. This supports the author's hypothesis that the inductive bias provided by the winning tickets is *dataset-independent*.

Second, we observe that winning tickets generated from more complex datasets (with higher number of classes) generalize better than those generated on relatively simpler datasets. Winning tickets generated on CIFAR-100 transfer better than those generated by those on CIFAR-10. This effect can be clearly seen in Figure 3 for ResNet50 architecture, while for VGG19 architecture both winning tickets show similar accuracy.

Third, while we observe that the winning rates transferred similarly for both ResNet50 and VGG19 architectures, the ResNet50 architecture tickets showed a sharper degradation in accuracy at higher pruning fractions compared to VGG19 architecture tickets. This can be observed by comparing Figure 3 with Figure 4.

We do not report the results of VGG19 winning tickets on FashionMNIST as the experiments for the same are still running and did not complete till report submission deadline.

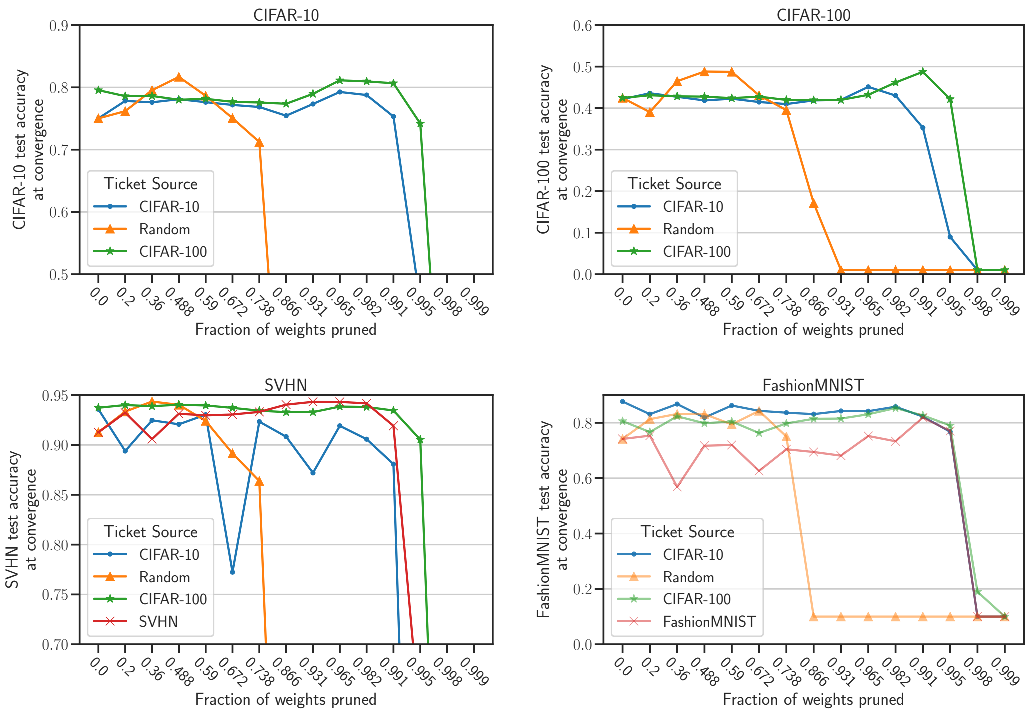

Figure 3: Transfer of winning ticket initializations across datasets with ResNet50

## 5.4 Discussion

We successfully replicate the experiments in the original paper. From our results, we observe that the winning ticket initializations transfer across multiple datasets and optimizers. This suggests that the winning tickets provide an inductive bias while training pruned models and are not overfitting a particular optimizer or dataset. We also observe that tickets over more complex datasets generalize better. Finally, we see that the different architectures have different sensitivity to pruning fractions, with ResNet50 showing sharper accuracy degradation at higher pruning fractions than VGG19. Overall, these observations motivate further work in area of neural network initializations. Further, as generating lottery tickets using iterative pruning is computationally expensive and time-consuming, more efficient methods for generating winning tickets are needed.

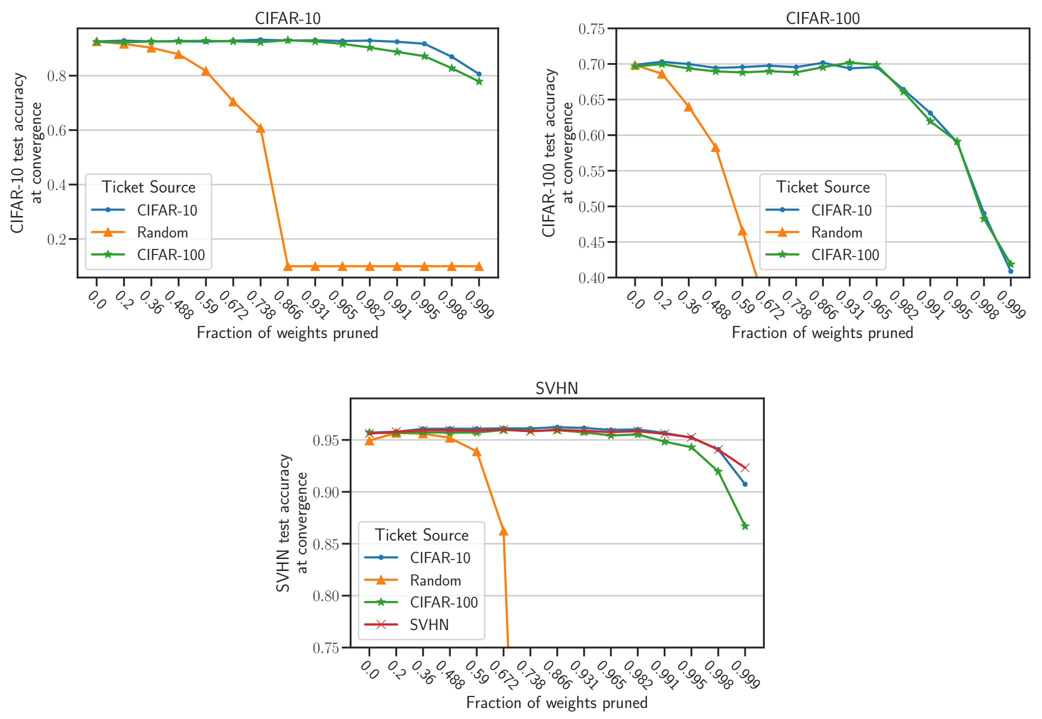

Figure 4: Transfer of winning ticket initializations across datasets with VGG19

## 6 Future Work

We plan to generate more winning tickets for these experiments with multiple random seeds to strengthen the reproducibility procedure we follow. If we are provided with access to more compute resources, we also plan to replicate the results presented on larger datasets like ImageNet and Places365. We plan to open-source new winning tickets as and when we generate them.

## 7 Conclusion

The original paper [4] investigates the generalizability of winning ticket initializations across datasets and optimizers. We replicate the experiments of the original paper from scratch. Our results support the major claims of the original paper and empirically answer the target questions. Our results empirically show that the winning ticket initializations can be transferred across datasets and optimizers. We appreciate the authors' ability to explain their experiments and observations in a lucid and replicable manner. While the results are replicable, we find that process of reproducing the results is extremely compute-intensive. Hence along with our code base, we also open-source the winning tickets we find during our experiments.

## 8 Acknowledgements

We would like to thank Ari Morcos, Facebook AI Research for resolving our queries throughout this work. We would also like to thank Prof. Anirban Dasgupta, IIT Gandhinagar and Prof. Nipun Batra, IIT Gandhinagar for providing us Google Cloud credits for running the experiments. We would also like to thank the organizers of NeurIPS Reproducibility Challenge 2019 for providing us with CodeOcean credits. Further we thank Mr. Rachit Chhaya, IIT Gandhinagar for reviewing this report.

## References

[1] S. Han, J. Pool, J. Tran, and W. J. Dally, “Learning both weights and connections for efficient neural networks,” *CoRR*, vol. abs/1506.02626, 2015.

[2] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, “Pruning filters for efficient convnets,” *CoRR*, vol. abs/1608.08710, 2016.

[3] J. Frankle and M. Carbin, “The lottery ticket hypothesis: Finding sparse, trainable neural networks,” in *International Conference on Learning Representations*, 2019.

[4] A. S. Morcos, H. Yu, M. Paganini, and Y. Tian, “One ticket to win them all: generalizing lottery ticket initializations across datasets and optimizers,” 2019.

[5] J. Frankle, G. K. Dziugaite, D. M. Roy, and M. Carbin, “The lottery ticket hypothesis at scale,” *CoRR*, vol. abs/1903.01611, 2019.

[6] A. Paszke, S. Gross, S. Chintala, G. Chanan, E. Yang, Z. DeVito, Z. Lin, A. Desmaison, L. Antiga, and A. Lerer, “Automatic differentiation in pytorch,” in *NIPS-W*, 2017.

[7] K. He, X. Zhang, S. Ren, and J. Sun, “Deep residual learning for image recognition,” *CoRR*, vol. abs/1512.03385, 2015.

[8] A. Krizhevsky, “Learning multiple layers of features from tiny images,” 2009.

[9] Y. Netzer, T. Wang, A. Coates, A. Bissacco, B. Wu, and A. Y. Ng, “Reading digits in natural images with unsupervised feature learning,” in *NIPS Workshop on Deep Learning and Unsupervised Feature Learning 2011*, 2011.

[10] H. Xiao, K. Rasul, and R. Vollgraf, “Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms,” 2017.

[11] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, “ImageNet: A Large-Scale Hierarchical Image Database,” in *CVPR09*, 2009.

[12] B. Zhou, A. Lapedriza, A. Khosla, A. Oliva, and A. Torralba, “Places: A 10 million image database for scene recognition,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2017.