Net-Trim: Convex Pruning of Deep Neural Networks with Performance Guarantee

Alireza Aghasi (Prev. IBM Research, Curr. GSU Business School), Afshin Abdi (Georgia Tech),
Nam Nguyen (IBM Research) & Justin Romberg (Georgia Tech)
aaghasi@gsu.edu

Abstract

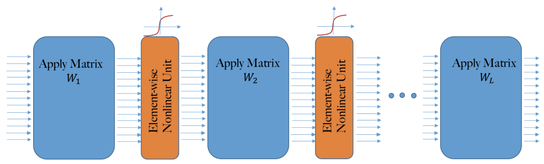
Presentation and analysis of a new technique for model reduction in deep neural networks:

- Our algorithm **prunes** (**sparsifies**) a trained network via **convex optimization** for networks with **ReLU** activation
- The program seeks a sparse set of weights at each layer that keeps the layer inputs and outputs consistent with the originally trained model
- We present both **parallel** and **cascade** versions of the algorithm and prove an overall network consistency with the originally trained network
- **Sample complexity**: we show that if the network response can be described using a maximum number of s non-zero weights per node, these weights can be learned from $\mathcal{O}(s \log N)$ samples
- We present an **ADMM scheme** to implement the convex pruning scheme

1 Introduction

– Linear models are not flexible enough to represent all systems

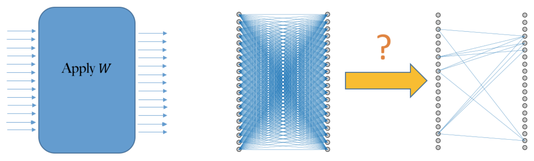
– Deep neural networks are nonlinear models, which are able to learn any system of arbitrary complexity!



– In linear models we aim to learn the matrix (or vector) $\boldsymbol{W}$, while in neural nets we aim to learn $\boldsymbol{W}_1, \cdots \boldsymbol{W}_L$.

– Each layer of a neural net can be represented by a graph

– An edge between two nodes is present when $W_{i,j} \neq 0$



– A similar model reduction as LASSO searches for sparse matrices $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L$

– We ideally want to address

$$\min_{\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L} \sum_{\ell=1}^{L} \|\boldsymbol{W}_\ell\|_1 \quad \text{subject to:} \quad \|F(\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L, \boldsymbol{X}) - \boldsymbol{Y}\| \leq \epsilon,$$

where $F(\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L, \boldsymbol{X}) = \boldsymbol{W}_L^T \sigma(\boldsymbol{W}_{L-1}^T \sigma(\cdots \sigma(\boldsymbol{W}_1^T \boldsymbol{X}) \cdots))$

– The resulting program is highly nonconvex and easily gets trapped in local minima

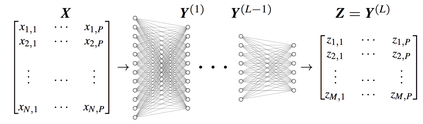
– Efforts to convexify the neural nets objective have failed and even some proposed convex approximations are not computationally tractable [1]

– Most pruning strategies are based on heuristics, e.g., ℓ_1 penalty, Dropout, DropConnect which rely on random removal of connections/activations [3], [4], compression algorithms which rely on thresholding and retraining [2]

– Due to the complexity of the model, there are only few works with mathematical guarantees about the neural nets performance

– None of the recent works with theoretical guarantees have addressed the pruning problem

2 Network Model & Net-Trim Pruning



- Input: P training samples $\boldsymbol{x}_p$, $p = 1, \cdots, P$, stacked up as: $\boldsymbol{X} = [\boldsymbol{x}_1, \cdots, \boldsymbol{x}_P]$
- Weight Matrices: passing through each layer

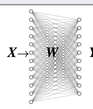
$$\boldsymbol{Y}^{(\ell)} = \max\left(\boldsymbol{W}_\ell^T \boldsymbol{Y}^{(\ell-1)}, 0\right), \qquad \ell = 1, \cdots, L, \quad \text{where} \quad \boldsymbol{Y}^{(0)} = \boldsymbol{X}, \quad \boldsymbol{Y}^{(L)} = \boldsymbol{Z}.$$

- Proper Scaling: since $\max(\alpha x, 0) = \alpha \max(x, 0)$ for $\alpha > 0$, we can normalize by:

$$\boldsymbol{W}_\ell \to \boldsymbol{W}_\ell / \|\boldsymbol{W}_\ell\|_1, \qquad \boldsymbol{Y}^{(\ell+1)} \to \boldsymbol{Y}^{(\ell+1)} / \prod_{j=0}^{\ell} \|\boldsymbol{W}_j\|_1.$$

Definition

A given neural network $\mathcal{TN}(\{\boldsymbol{W}_\ell\}_{\ell=1}^L, \boldsymbol{X})$ is link-normalized when $\|\boldsymbol{W}_\ell\|_1 = 1$ for every layer $\ell = 1, \cdots, L$.



$$\boldsymbol{Y} = \max\left(\boldsymbol{W}^T \boldsymbol{X}, 0\right)$$

- Inspired by the ℓ_1 recovery problem we consider the following program to prune a layer

$$\hat{\boldsymbol{W}} = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \left\|\max\left(\boldsymbol{U}^T \boldsymbol{X}, 0\right) - \boldsymbol{Y}\right\|_F \leq \epsilon.$$

To convexify: knowing that $y_{m,p}$ is either zero or positive, we enforce the $\max(.,0)$ argument to be negative when $y_{m,p} = 0$, and close to $y_{m,p}$ elsewhere:

$$\begin{cases} \sum\limits_{m,p:\ y_{m,p}>0} \left(\boldsymbol{u}_m^T \boldsymbol{x}_p - y_{m,p}\right)^2 \leq \epsilon^2 \\ \boldsymbol{u}_m^T \boldsymbol{x}_p \leq v_{m,p} \qquad \text{for } m,p:\ y_{m,p} = 0 \end{cases} \iff \boldsymbol{U} \in \mathcal{C}_\epsilon(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{V}).$$

$$\hat{\boldsymbol{W}} = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_\epsilon(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{0}).$$

Algorithm 1 Parallel Net-Trim

Input: $\boldsymbol{X}$, $\epsilon > 0$, and link-normalized $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L$
$\boldsymbol{Y}^{(0)} \leftarrow \boldsymbol{X}$
for $\ell = 1, \cdots, L$ **do**
  $\boldsymbol{Y}^{(\ell)} \leftarrow \max\left(\boldsymbol{W}_\ell^T \boldsymbol{Y}^{(\ell-1)}, 0\right)$
end for
for all $\ell = 1, \cdots, L$ **do**
  $\hat{\boldsymbol{W}}_\ell \leftarrow \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_\epsilon\left(\boldsymbol{Y}^{(\ell-1)}, \boldsymbol{Y}^{(\ell)}, \boldsymbol{0}\right)$ % retraining
end for
Output: $\hat{\boldsymbol{W}}_1, \cdots, \hat{\boldsymbol{W}}_L$

Theorem

Given a link-normalized network $\mathcal{TN}(\{\boldsymbol{W}_\ell\}_{\ell=1}^L, \boldsymbol{X})$ with layer outcomes $\boldsymbol{Y}^{(\ell)}$, consider retraining each layer individually via

$$\hat{\boldsymbol{W}}_\ell = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_\ell}\left(\boldsymbol{Y}^{(\ell-1)}, \boldsymbol{Y}^{(\ell)}, \boldsymbol{0}\right).$$

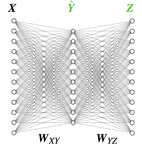
For the retrained network $\mathcal{TN}(\{\hat{\boldsymbol{W}}_\ell\}_{\ell=1}^L, \boldsymbol{X})$ with layer outcomes $\hat{\boldsymbol{Y}}^{(\ell)} = \max(\hat{\boldsymbol{W}}_\ell^T \hat{\boldsymbol{Y}}^{(\ell-1)}, 0)$,

$$\left\|\hat{\boldsymbol{Y}}^{(\ell)} - \boldsymbol{Y}^{(\ell)}\right\|_F \leq \sum_{j=1}^{\ell} \epsilon_j.$$

corollary

Using Algorithm 1 ($\epsilon_1 = \cdots = \epsilon_L = \epsilon$), the ultimate network outcome obeys

$$\left\|\hat{\boldsymbol{Y}}^{(L)} - \boldsymbol{Y}^{(L)}\right\|_F \leq L\epsilon.$$



Cascade Scheme:

$$\hat{\boldsymbol{W}}_{XY} = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_\epsilon(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{0})$$

$$\hat{\boldsymbol{Y}} = \max(\hat{\boldsymbol{W}}_{XY}^T \boldsymbol{X}, 0)$$

$$\hat{\boldsymbol{W}}_{YZ} = \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_2}(\hat{\boldsymbol{Y}}, \boldsymbol{Z}, \boldsymbol{V})$$

To retrain the second layer, we ideally expect the following program

$$\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_\epsilon(\hat{\boldsymbol{Y}}, \boldsymbol{Z}, \boldsymbol{0}).$$

However, there is no faesibility guarantee, i.e., there exists $\boldsymbol{W} = [\boldsymbol{w}_1, \cdots, \boldsymbol{w}_{N_2}]$ such that

$$\begin{cases} \sum\limits_{m,p:\ z_{m,p}>0} \left(\boldsymbol{w}_m^T \hat{\boldsymbol{y}}_p - z_{m,p}\right)^2 \leq \epsilon^2 \\ \boldsymbol{w}_m^T \hat{\boldsymbol{y}}_p \leq 0 \qquad \text{for } m,p:\ z_{m,p} = 0 \end{cases}.$$

If instead of $\hat{\boldsymbol{Y}}$ the program was parameterized by $\boldsymbol{Y}$, a natural feasible point would have been $\boldsymbol{W}_{YZ}$. So we can plug in $\boldsymbol{W}_{YZ}$ and slack the constraints to warrant feasibility.

We address the slacked program: $\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_{\epsilon_2}(\hat{\boldsymbol{Y}}, \boldsymbol{Z}, \boldsymbol{V})$.
Here, $\boldsymbol{V} = \boldsymbol{W}_{YZ}^T \hat{\boldsymbol{Y}}$ and

$$\epsilon_2 = \gamma \sum_{m,p:\ z_{m,p}>0} \left(\boldsymbol{w}_{YZ,m}^T \hat{\boldsymbol{y}}_p - z_{m,p}\right)^2.$$

The constants $\gamma \geq 1$ (referred to as the *inflation rate*) is a free parameter, which controls the sparsity level.

Algorithm 2 Cascade Net-Trim

Input: $\boldsymbol{X}$, $\epsilon > 0$, $\gamma > 1$ and link-normalized $\boldsymbol{W}_1, \cdots, \boldsymbol{W}_L$
$\boldsymbol{Y} \leftarrow \max\left(\boldsymbol{W}_1^T \boldsymbol{X}, 0\right)$
$\hat{\boldsymbol{W}}_1 \leftarrow \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_\epsilon(\boldsymbol{X}, \boldsymbol{Y}, \boldsymbol{0})$
$\hat{\boldsymbol{Y}} \leftarrow \max(\hat{\boldsymbol{W}}_1^T \boldsymbol{X}, 0)$
for $\ell = 2, \cdots, L$ **do**
  $\boldsymbol{Y} \leftarrow \max(\boldsymbol{W}_\ell^T \boldsymbol{Y}, 0)$
  $\epsilon \leftarrow (\gamma \sum_{m,p:\ z_{m,p}>0} (\boldsymbol{w}_{\ell,m}^T \hat{\boldsymbol{y}}_p - y_{m,p})^2)^{1/2}$ % $\boldsymbol{w}_{\ell,m}$ is the m-th column of $\boldsymbol{W}_\ell$
  $\hat{\boldsymbol{W}}_\ell \leftarrow \arg\min_{\boldsymbol{U}} \ \|\boldsymbol{U}\|_1 \quad s.t. \quad \boldsymbol{U} \in \mathcal{C}_\epsilon(\hat{\boldsymbol{Y}}, \boldsymbol{Y}, \boldsymbol{W}_\ell^T \hat{\boldsymbol{Y}})$
  $\hat{\boldsymbol{Y}} \leftarrow \max(\hat{\boldsymbol{W}}_\ell^T \hat{\boldsymbol{Y}}, 0)$
end for
Output: $\hat{\boldsymbol{W}}_1, \cdots, \hat{\boldsymbol{W}}_L$

Theorem

Given a link-normalized network $\mathcal{TN}(\{\boldsymbol{W}_\ell\}_{\ell=1}^L, \boldsymbol{X})$ with layer outcomes $\boldsymbol{Y}^{(\ell)}$, consider a cascade retraining where

$$\epsilon_\ell^2 = \gamma_\ell \sum_{m,p:\ y_{m,p}^{(\ell)}>0} \left(\boldsymbol{w}_{\ell,m}^T \hat{\boldsymbol{y}}_p^{(\ell-1)} - y_{m,p}^{(\ell)}\right)^2,$$

$\hat{\boldsymbol{Y}}^{(\ell)} = \max(\hat{\boldsymbol{W}}_\ell^T \hat{\boldsymbol{Y}}^{(\ell-1)}, 0)$, $\hat{\boldsymbol{Y}}^{(1)} = \max(\hat{\boldsymbol{W}}_1^T \boldsymbol{X}, 0)$ and $\gamma_\ell > 1$. For $\mathcal{TN}(\{\hat{\boldsymbol{W}}_\ell\}_{\ell=1}^L, \boldsymbol{X})$ being the retrained network

$$\left\|\hat{\boldsymbol{Y}}^{(\ell)} - \boldsymbol{Y}^{(\ell)}\right\|_F \leq \epsilon_1 \sqrt{\prod_{j=2}^{\ell} \gamma_j}.$$

Corollary

Using Algorithm 2, the ultimate network outcome obeys

$$\left\|\hat{\boldsymbol{Y}}^{(L)} - \boldsymbol{Y}^{(L)}\right\|_F \leq \gamma^{\frac{(L-1)}{2}} \epsilon.$$

3 Sample Complexity

- **Key question**: if there exists a sparse transormation matrix relating the input and output of a layer, how many samples are required to learn such model.
- **Interested in the under-determined case** ($\epsilon = 0$): the relation between the input and output can be established via more than one $\boldsymbol{W}$.

$$(\epsilon = 0): \quad \min_{\boldsymbol{W}} \ \|\boldsymbol{W}\|_1 \quad s.t. \quad \begin{cases} \boldsymbol{w}_m^T \boldsymbol{x}_p = y_{m,p} & \text{for } m,p:\ y_{m,p} > 0 \\ \boldsymbol{w}_m^T \boldsymbol{x}_p \leq 0 & \text{for } m,p:\ y_{m,p} = 0 \end{cases},$$

The program decouples and can be separately written for each column of $\boldsymbol{W}$ denoted as $\boldsymbol{w}$:

$$\min_{\boldsymbol{w}} \ \|\boldsymbol{w}\|_1 \quad s.t. \quad \begin{cases} \boldsymbol{X}_{:,\Omega}^T \boldsymbol{w} = \boldsymbol{y}_\Omega \\ \boldsymbol{X}_{:,\Omega^c}^T \boldsymbol{w} \preceq \boldsymbol{0} \end{cases}, \quad \text{where} \quad \Omega = \{p : y_p > 0\},$$

or in block form:

$$\min_{\boldsymbol{w}, \boldsymbol{s}} \ \|\boldsymbol{w}\|_1 \quad s.t. \quad \tilde{\boldsymbol{X}} \begin{bmatrix} \boldsymbol{w} \\ \boldsymbol{s} \end{bmatrix} = \boldsymbol{y}, \quad \boldsymbol{s} \preceq \boldsymbol{0}, \tag{1}$$

where

$$\tilde{\boldsymbol{X}} = \begin{bmatrix} \boldsymbol{X}_{:,\Omega}^T & \boldsymbol{0} \\ \boldsymbol{X}_{:,\Omega^c}^T & -\boldsymbol{I} \end{bmatrix} \quad \text{and} \quad \boldsymbol{y} = \begin{bmatrix} \boldsymbol{y}_\Omega \\ \boldsymbol{0} \end{bmatrix}.$$

Proposition

Consider a pair $(\boldsymbol{w}^*, \boldsymbol{s}^*) \in (\mathbb{R}^{n_1}, \mathbb{R}^{n_2})$, which is feasible for the convex program (31). If there exists a vector $\boldsymbol{\Lambda} = [\Lambda_\ell] \in \mathbb{R}^{n_1+n_2}$ in the range of the sensing matrix $\tilde{\boldsymbol{X}}^T$ with entries satisfying

$$\begin{cases} -1 < \Lambda_\ell < 1 & \ell \in \text{supp}^c\, \boldsymbol{w}^* \\ 0 < \Lambda_{n_1+\ell} & \ell \in \text{supp}^c\, \boldsymbol{s}^* \end{cases}, \quad \begin{cases} \Lambda_\ell = \text{sign}(w_\ell^*) & \ell \in \text{supp}\, \boldsymbol{w}^* \\ \Lambda_{n_1+\ell} = 0 & \ell \in \text{supp}\, \boldsymbol{s}^* \end{cases},$$

and for $\tilde{\Gamma} = \text{supp}\, \boldsymbol{w}^* \cup \{n_1 + \text{supp}\, \boldsymbol{s}^*\}$ the restricted matrix $\tilde{\boldsymbol{X}}_{:,\tilde{\Gamma}}$ is full column rank, then the pair $(\boldsymbol{w}^*, \boldsymbol{s}^*)$ is the unique solution to (31).

Theorem

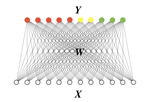
Let $\boldsymbol{w}^* \in \mathbb{R}^N$ be an arbitrary s-sparse vector, $\boldsymbol{X} \in \mathbb{R}^{N \times P}$ a Gaussian matrix representing the samples and $\mu > 1$ a fixed value. Given $P = (11s+7)\mu \log N$ observations of the type $\boldsymbol{y} = \max(\boldsymbol{X}^T \boldsymbol{w}^*, 0)$, with probability exceeding $1 - N^{1-\mu}$ the vector $\boldsymbol{w}^*$ can be learned exactly through the proposed convex model.

Corollary

Consider an arbitrary matrix $\boldsymbol{W}^* = [\boldsymbol{w}_1^*, \cdots, \boldsymbol{w}_M^*] \in \mathbb{R}^{N \times M}$, where $s_m = \|\boldsymbol{w}_m^*\|_0$, and $0 < s_m \leq s_{\max}$ for $m = 1, \cdots, M$. For $\boldsymbol{X} \in \mathbb{R}^{N \times P}$ being a Gaussian matrix, set $\boldsymbol{Y} = \max(\boldsymbol{W}^{*T} \boldsymbol{X}, 0)$. If $\mu > (1 + \log_N M)$ and $P = (11 s_{\max} + 7)\mu \log N$, for $\epsilon = 0$, $\boldsymbol{W}^*$ can be accurately learned through (1) with probability exceeding $1 - \sum_{m=1}^{M} N^{1-\mu \frac{11 s_m + 7}{11 s_m + 7}}$.

4 Additional Remarks & Experiments

– For each layer the retraining can be performed on separate clusters of the output nodes in parallel



– The $\mathcal{O}(s \log N/s)$ sample complexity is generalizable to all the layers under certain well conditioning of the input covariance matrix

– We have been able to present an **ADMM** scheme to address the Net-Trim core program (the details of the program are available in the paper and the **code is available online**)

In the following example, we classify 200 points in the 2D plane with a network of two hidden layers (200 neurons each) and Net-Trim generates a 93% smaller network with similar performance

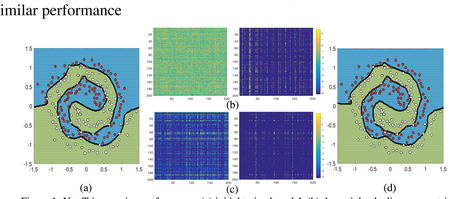


Figure 1: Net-Trim pruning performance; (a) initial trained model; (b) the weighted adjacency matrix relating the two hidden layers before (left) and after (right) the application of Net-Trim; (c) left: the adjacency matrix after training the network with Dropout and ℓ_1 regularization; right: after retraining via Net-Trim; (d) the retrained classifier

The following experiments compare Net-Trim against HPTD in [2]

Table 1: The test accuracy of different models before and after Net-Trim (NT) and HPTD [14]. Without a fine training (FT) step, Net-Trim produces pruned networks in the majority of cases more accurate than HPTD and with no risk of poor local minima. Adding an additional FT step makes Net-Trim consistently prominent

	NN2-10K	NN3-30K	NN3-60K	NN4-10K	NN4-30K	NN4-60K
Init. Mod. Acc. (%)	95.59	97.58	98.18	98.37	99.11	99.25
Total Pruning (%)	40.86	30.69	29.38	43.91	39.11	45.74
NT Overall Disc. (%)	1.98	1.31	1.77	1.22	0.75	0.55
NT No FT Acc. (%)	95.47	97.55	98.1	98.31	99.15	99.25
HPTD No FT Acc. (%)	9.3	10.34	8.92	19.17	55.92	30.17
NT + FT Acc. (%)	95.85	97.67	98.12	98.35	99.21	99.33
HPTD + FT Acc. (%)	93.56	97.32	EMT	98.16	EMT	EMT

	NN2-10K	NN3-30K	NN3-60K	NN4-10K	NN4-30K	NN4-60K
Init. Mod. Acc. (%)	95.59	97.58	98.18	98.37	99.11	99.25
Total Pruning (%)	75.87	75.82	77.40	76.18	77.63	81.62
NT Overall Disc. (%)	4.95	11.01	11.47	3.65	5.32	8.93
NT No FT Acc. (%)	94.92	95.97	97.35	97.91	99.08	98.96
HPTD No FT Acc. (%)	8.97	10.1	8.92	31.18	73.36	46.84
NT + FT Acc. (%)	95.89	97.69	98.19	98.40	99.17	99.26
HPTD + FT Acc. (%)	95.61	EMT	97.96	EMT	99.01	99.06

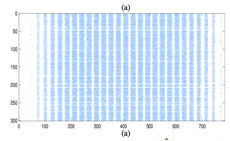


Figure 2: Visualization of $\hat{\boldsymbol{W}}_1$ in NN3-60K; (a) Net-Trim output; (b) standard HPTD

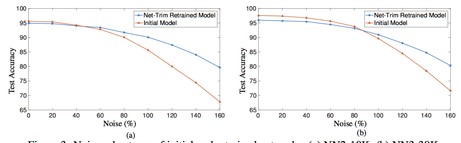


Figure 3: Noise robustness of initial and retrained networks; (a) NN2-10K; (b) NN3-30K

References

[1] F. Bach. Breaking the curse of dimensionality with convex neural networks. *Technical report*, 2014.

[2] S. Han, J. Pool, J. Tran, and W. Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

[3] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

[4] L. Wan, M. Zeiler, S. Zhang, Y. LeCun, and R. Fergus. Regularization of neural networks using dropconnect. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.