Understanding the Disharmony between Dropout and Batch Normalization by Variance Shift

Xiang Li [1] **Shuo Chen** [1] **Xiaolin Hu** [2] **Jian Yang** [1]

Abstract

This paper first answers the question "why do the two most powerful techniques Dropout and Batch Normalization (BN) often lead to a worse performance when they are combined together?" in both theoretical and statistical aspects. Theoretically, we find that Dropout would shift the variance of a specific neural unit when we transfer the state of that network from train to test. However, BN would maintain its statistical variance, which is accumulated from the entire learning procedure, in the test phase. The inconsistency of that variance (we name this scheme as "variance shift") causes the unstable numerical behavior in inference that leads to more erroneous predictions finally, when applying Dropout before BN. Thorough experiments on DenseNet, ResNet, ResNeXt and Wide ResNet confirm our findings. According to the uncovered mechanism, we next explore several strategies that modifies Dropout and try to overcome the limitations of their combination by avoiding the variance shift risks.

1. Introduction

(Srivastava et al., 2014) brought Dropout as a simple way to prevent neural networks from overfitting. It has been proved to be significantly effective over a large range of machine learning areas, such as image classification (Szegedy et al., 2015), speech recognition (Hannun et al., 2014) and even natural language processing (Kim et al., 2016). Before the birth of Batch Normalization, it became a necessity of almost all the state-of-the-art networks and successfully boosted their performances against overfitting risks, despite its amazing simplicity.

(Ioffe & Szegedy, 2015) demonstrated Batch Normalization (BN), a powerful skill that not only speeded up all the modern architectures but also improved upon their strong baselines by acting as regularizers. Therefore, BN has been implemented in nearly all the recent network structures (Szegedy et al., 2016; 2017; Howard et al., 2017; Zhang et al., 2017) and demonstrates its great practicability and effectiveness.

However, the above two nuclear weapons always fail to obtain an extra reward when combined together practically. In fact, a network even performs worse and unsatisfactorily when it is equipped with BN and Dropout simultaneously. (Ioffe & Szegedy, 2015) have already realized that BN eliminates the need for Dropout in some cases – the authors exposed the incompatibility between them, thus conjectured

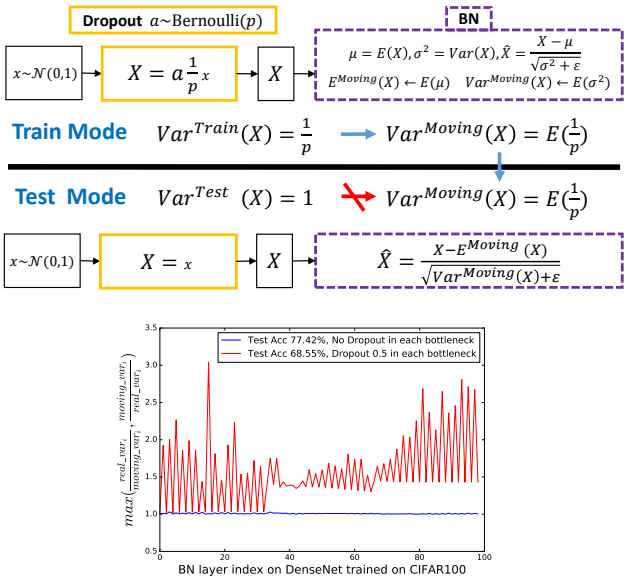

Figure 1. **Up:** a simplified mathematical illustration of "variance shift". In test mode, the neural variance of X is different from that in train mode caused by Dropout, yet BN attempts to regard that variance as the popular statistic accumulated from training. Note that p denotes for the Dropout retain ratio and a comes from Bernoulli distribution which has probability p of being 1. **Down:** variance shift in experimental statistics on DenseNet trained on CIFAR100 dataset. The curves are both calculated from the *same training data*. "$moving\_var_i$" is the moving variance (take its mean value instead if it's a vector) that the i-th BN layer accumulates during the entire learning, and "$real\_var_i$" stands for the real variance of neural response before the i-th BN layer in inference.

[1]DeepInsight@PCALab, Nanjing University of Science and Technology, China [2]Tsinghua National Laboratory for Information Science and Technology (TNList) Department of Computer Science and Technology, Tsinghua University, China. Correspondence to: Xiang Li <xiang.li.implus@njust.edu.cn>.

that BN provides similar regularization benefits as Dropout intuitively. More evidences are provided in the modern architectures such as ResNet (He et al., 2016a;b), ResNeXt (Xie et al., 2017), DenseNet (Huang et al., 2016), where the best performances are all obtained by BN with the absence of Dropout. Interestingly, a recent study Wide ResNet (WRN) (Zagoruyko & Komodakis, 2016) show that it is positive for Dropout to be applied in the WRN design with a large feature dimension. So far, previous clues leave us a mystery about the confusing and complicated relationship between Dropout and BN. Why do they conflict in most of the common architectures? Why do they cooperate friendly sometimes as in WRN?

We discover the key to understand the disharmony between Dropout and BN is the inconsistent behaviors of neural variance during the switch of networks' state. Considering one neural response X as illustrated in Figure 1, when the state changes from train to test, Dropout would scale the response by its Dropout retain ratio (i.e. p) that actually changes the neural variance as in learning, yet BN still maintains its statistical moving variance of X. This mismatch of variance could lead to a numerical instability (see red curve in Figure 1). As the signals go deeper, the numerical deviation on the final predictions may amplify, which drops the system's peformance. We name this scheme as "variance shift" for simplicity. Instead, without Dropout, the real neural variances in inference would appear very closely to the moving ones accumulated by BN (see blue curve in Figure 1), which is also preserved with a higher test accuracy.

Theoretically, we deduced the "variance shift" under two general conditions, and found a satisfied explanation for the aforementioned mystery between Dropout and BN. Further, a large range of experimental statistics from four modern networks (i.e., PreResNet (He et al., 2016b), ResNeXt (Xie et al., 2017), DenseNet (Huang et al., 2016), Wide ResNet (Zagoruyko & Komodakis, 2016)) on the CIFAR10/100 datasets verified our findings as expected.

Since the central reason for their performance drop was discovered, we adopted two strategies that explored the possibilities to overcome the limitation of their combination. One was to apply Dropout after all BN layers and another was to modify the formula of Dropout and made it less sensitive to variance. By avoiding the variance shift risks, most of them worked well and achieved extra improvements.

2. Related Work and Preliminaries

Dropout (Srivastava et al., 2014) can be interpreted as a way of regularizing a neural network by adding noise to its hidden units. Specifically, it involves multiplying hidden activations by Bernoulli distributed random variables which take the value 1 with probability p ($0 \leq p \leq 1$) and 0 otherwise[1]. Importantly, the test scheme is quite different from the train. During training, the information flow goes through the dynamic sub-network. At test time, the neural responses are scaled by the Dropout retain ratio, in order to approximate an equally weighted geometric mean of the predictions of an exponential number of learned models that share parameters. Consider a feature vector $\mathbf{x} = (x_1 \dots x_d)$ with channel dimension d. Note that this vector could be a part (one location) of convolutional feature-map or the output of the fully connected layer, i.e., it doesnot matter which type of network it lies in. If we apply Dropout on $\mathbf{x}$, for one unit $x_k, k = 1 \dots d$, in the train phase, it is:

$$\widehat{x}_k = a_k x_k, \tag{1}$$

where $a_k \sim P$ that comes from the Bernoulli distribution:

$$P(a_k) = \begin{cases} 1-p, & a_k = 0 \\ p, & a_k = 1 \end{cases}, \tag{2}$$

and $\mathbf{a} = (a_1 \dots a_d)$ is a vector of *independent* Bernoulli random variables. At test time for Dropout, one should scale down the weights by multiplying them by a factor of p. As introduced in (Srivastava et al., 2014), another way to achieve the same effect is to scale up the retained activations by multiplying by $\frac{1}{p}$ at training time and not modifying the weights at test time. It is more popular on practical implementations, thus we employ this formula of Dropout in both analyses and experiments. Therefore, the hidden activation in the train phase would be:

$$\widehat{x}_k = a_k \frac{1}{p} x_k, \tag{3}$$

whilst in inference it would be simple like: $\widehat{x}_k = x_k$.

Batch Normalization (BN) (Ioffe & Szegedy, 2015) proposes a deterministic information flow by normalizing each neuron into zero mean and unit variance. Considering values of x (for clarity, $x \equiv x_k$) over a mini-batch: $\mathcal{B} = \{x^{(1)\dots(m)}\}$[2] with m instances, we have the form of "normalize" part:

$$\mu = \frac{1}{m}\sum_{i=1}^{m} x^{(i)}, \sigma^2 = \frac{1}{m}\sum_{i=1}^{m}(x^{(i)} - \mu)^2, \widehat{x}^{(i)} = \frac{x^{(i)} - \mu}{\sqrt{\sigma^2 + \epsilon}}, \tag{4}$$

where μ and σ^2 would participate in the backpropagation. The normalization of activations that depends on the mini-batch allows efficient training, but is neither necessary nor desirable during inference. Therefore, BN accumulates the moving averages of neural means and variances during learning to track the accuracy of a model as it trains:

$$E^{Moving}(x) \leftarrow E_{\mathcal{B}}(\mu), Var^{Moving}(x) \leftarrow E'_{\mathcal{B}}(\sigma^2), \tag{5}$$

[1] p denotes for the Dropout retain ratio and $(1-p)$ denotes for the drop ratio in this paper.

[2] Note that we donot consider the "scale and shift" part in BN because the key of "variance shift" exists in its "normalize" part.

where $E_{\mathcal{B}}(\mu)$ denotes for the expectation of μ from multiple training mini-batches $\mathcal{B}$ and $E'_{\mathcal{B}}(\sigma^2)$ denotes for the expectation of the unbiased variance estimate (i.e., $\frac{m}{m-1} \cdot E_{\mathcal{B}}(\sigma^2)$) over multiple training mini-batches. They are all obtained by implementations of moving averages (Ioffe & Szegedy, 2015) and are fixed for linear transform during inference:

$$\widehat{x} = \frac{x - E^{Moving}(x)}{\sqrt{Var^{Moving}(x) + \epsilon}}. \tag{6}$$

3. Theoretical Analyses

From the preliminaries, one could notice that Dropout only ensures an "equally weighted geometric **mean** of the predictions of an exponential number of learned models" by the approximation from its test policy, as introduced in the original paper (Srivastava et al., 2014). This scheme poses the variance of the hidden units unexplored in a Dropout model. Therefore, the central idea is to investigate the variance of the neural response before a BN layer, where the Dropout is previously applied. This could be attributed into two cases generally, as shown in Figure 2. In case (a), the BN layer is directly subsequent to the Dropout layer and we only need to consider one neural response $X = a_k \frac{1}{p} x_k, k = 1 \dots d$ in train phase and $X = x_k$ in test phase. In case (b), the feature vector $\mathbf{x} = (x_1 \dots x_d)$ would be passed into a convolutional layer (or a fully connected layer) to form the neural response X. We also regard its corresponding weights (the convolutional filter or the fully connected weight) to be $\mathbf{w} = (w_1 \dots w_d)$, hence we get $X = \sum_{i=1}^{d} w_i a_i \frac{1}{p} x_i$ for learning and $X = \sum_{i=1}^{d} w_i x_i$ for test. For the ease of deduction, we assume that the inputs all come from the distribution with c mean and v variance (i.e., $E(x_i) = c, Var(x_i) = v, i = 1 \dots d, v > 0$) and we also start by studying the linear regime. We let the a_i and x_i be mutually independent, considering the property of Dropout. Due to the aforementioned definition, a_i and a_j are mutually independent as well.

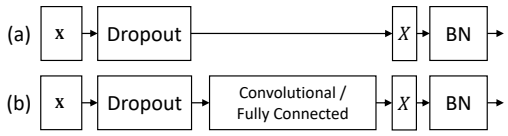

Figure 2. Two cases for analyzing variance shift.

Figure 2 (a)

Following the paradigms above, we have $Var^{Train}(X)$ as:

$$\begin{aligned} Var^{Train}(X) &= Var(a_k \frac{1}{p} x_k) = E((a_k \frac{1}{p} x_k)^2) - E^2(a_k \frac{1}{p} x_k) \\ &= \frac{1}{p^2} E(a_k^2) E(x_k^2) - \frac{1}{p^2} (E(a_k) E(x_k))^2 = \frac{1}{p}(c^2 + v) - c^2 \end{aligned} \tag{7}$$

In inference, BN keeps the moving average of variance (i.e., $E'_{\mathcal{B}}(\frac{1}{p}(c^2+v)-c^2)$) fixed. In another word, BN wishes that the variance of neural response X, which comes from the input images, is supposed to be close to $E'_{\mathcal{B}}(\frac{1}{p}(c^2+v)-c^2)$. However, Dropout breaks the harmony when it comes to its test stage by having $X = x_k$ to get $Var^{Test}(X) = Var(x_k) = v$. If putting $Var^{Test}(X)$ into the unbiased variance estimate, it would become $E'_{\mathcal{B}}(v)$ which is obviously different from the popular statistic $E'_{\mathcal{B}}(\frac{1}{p}(c^2+v)-c^2)$ of BN during training when Dropout ($p < 1$) is applied. Therefore, the shift ratio is obtained:

$$\triangle(p) = \frac{Var^{Test}(X)}{Var^{Train}(X)} = \frac{v}{\frac{1}{p}(c^2 + v) - c^2} \tag{8}$$

In case (a), the variance shift happens via a coefficient $\triangle(p) \leq 1$. Since modern neural networks carry a deep feedforward topologic structure, previous deviate numerical manipulations could lead to more uncontrollable numerical outputs of subsequent layers (Figure 1). It brings the chain reaction of amplified shift of variances (even affects the means further) in every BN layers sequentially as the networks go deeper. We would show that it directly leads to a dislocation of final predictions and makes the system suffer from a performance drop later in the statistical experimental part (e.g., Figure 4, 6 in Section 4).

In this design (i.e., BN directly follows Dropout), if we want to alleviate the variance shift risks, i.e., $\triangle(p) \to 1$, the only thing we can do is to eliminate Dropout and set the Dropout retain ratio $p \to 1$. Fortunately, the architectures where Dropout brings benefits (e.g., in Wide ResNet) donot follow this type of arrangement. In fact, they adopt the case (b) in Figure 2, which is more common in practice, and we would describe it in details next.

Figure 2 (b)

At this time, X would be obtained by $\sum_{i=1}^{d} w_i a_i \frac{1}{p} x_i$, where $\mathbf{w}$ denotes for the corresponding weights that act on the feature vector $\mathbf{x}$, along with the Dropout applied. For the ease of deduction, we assume that in the very later epoch of training, the weights of $\mathbf{w}$ remains constant given the gradients become significantly close to zero. Similarly, we can write $Var^{Train}(X)$ by following the formula of variance:

$$\begin{aligned} Var^{Train}(X) &= Cov(\sum_{i=1}^{d} w_i a_i \frac{1}{p} x_i, \sum_{i=1}^{d} w_i a_i \frac{1}{p} x_i) \\ &= \frac{1}{p^2} \sum_{i=1}^{d} (w_i)^2 Var(a_i x_i) \\ &+ \frac{1}{p^2} \sum_{i=1}^{d} \sum_{j \neq i}^{d} \rho_{i,j}^{ax} w_i w_j \sqrt{Var(a_i x_i)} \sqrt{Var(a_j x_j)} \\ &= (\frac{1}{p}(c^2 + v) - c^2)(\sum_{i=1}^{d} w_i^2 + \rho^{ax} \sum_{i=1}^{d} \sum_{j \neq i}^{d} w_i w_j), \end{aligned} \tag{9}$$

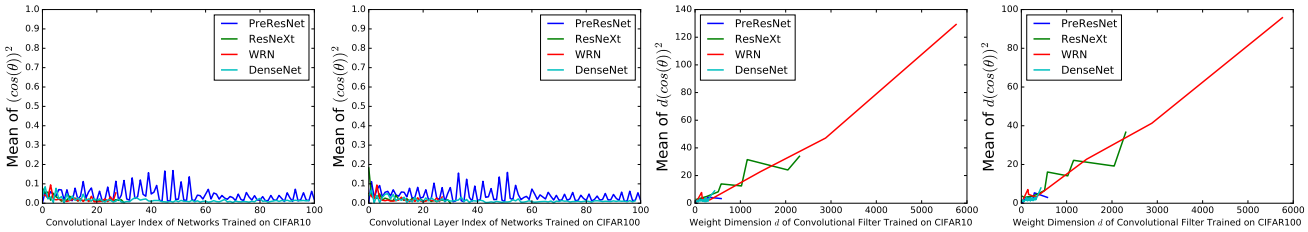

Figure 3. Statistical mean values of $(\cos\theta)^2$ and $d(\cos\theta)^2$. These four modern architectures are trained without Dropout on CIFAR10 and CIFAR100 respectively. We observe that $(\cos\theta)^2$ lies in $(0.01, 0.10)$ approximately in every network structure and various datasets. Interestingly, the term $d(\cos\theta)^2$ in WRN is significantly bigger than those on other networks mainly due to its larger channel width d.

where $\rho^{ax}_{i,j} = \frac{Cov(a_i x_i, a_j x_j)}{\sqrt{Var(a_i x_i)}\sqrt{Var(a_j x_j)}} \in [-1, 1]$. For the ease of deduction, we simplify all the linear correlation coefficients to be the same as a constant $\rho^{ax} \cong \rho^{ax}_{i,j}, \forall i, j = 1 \dots d, i \neq j$. Similarly, $Var^{Test}(X)$ is obtained:

$$\begin{aligned} Var^{Test}(X) &= Var(\sum_{i=1}^{d} w_i x_i) = Cov(\sum_{i=1}^{d} w_i x_i, \sum_{i=1}^{d} w_i x_i) \\ &= \sum_{i=1}^{d} w_i^2 v + \sum_{i=1}^{d}\sum_{j\neq i}^{d} \rho^x_{i,j} w_i w_j \sqrt{v}\sqrt{v} \\ &= v(\sum_{i=1}^{d} w_i^2 + \rho^x \sum_{i=1}^{d}\sum_{j\neq i}^{d} w_i w_j), \end{aligned} \tag{10}$$

where $\rho^x_{i,j} = \frac{Cov(x_i, x_j)}{\sqrt{Var(x_i)}\sqrt{Var(x_j)}} \in [-1, 1]$, and we also have a constant $\rho^x \cong \rho^x_{i,j}, \forall i, j = 1 \dots d, i \neq j$. Since a_i and x_i, a_i and a_j are mutually independent, we can get the relationship between ρ^{ax} and ρ^x:

$$\begin{aligned} \rho^{ax} \cong \rho^{ax}_{i,j} &= \frac{Cov(a_i x_i, a_j x_j)}{\sqrt{Var(a_i x_i)}\sqrt{Var(a_j x_j)}} \\ &= \frac{p^2 Cov(x_i, x_j)}{\frac{p(c^2+v)-p^2c^2}{v}\sqrt{Var(x_i)}\sqrt{Var(x_j)}} \\ &= \frac{v}{\frac{1}{p}(c^2+v) - c^2}\rho^x_{i,j} \cong \frac{v}{\frac{1}{p}(c^2+v) - c^2}\rho^x. \end{aligned} \tag{11}$$

According to Equation (9), (10) and (11), we can write the variance shift $\frac{Var^{Test}(X)}{Var^{Train}(X)}$ as:

$$\begin{aligned} &\frac{v(\sum_{i=1}^{d} w_i^2 + \rho^x \sum_{i=1}^{d}\sum_{j\neq i}^{d} w_i w_j)}{(\frac{1}{p}(c^2+v) - c^2)(\sum_{i=1}^{d} w_i^2 + \rho^{ax} \sum_{i=1}^{d}\sum_{j\neq i}^{d} w_i w_j)} \\ &= \frac{v\sum_{i=1}^{d} w_i^2 + v\rho^x \sum_{i=1}^{d}\sum_{j\neq i}^{d} w_i w_j}{(\frac{1}{p}(c^2+v) - c^2)\sum_{i=1}^{d} w_i^2 + v\rho^x \sum_{i=1}^{d}\sum_{j\neq i}^{d} w_i w_j} \\ &= \frac{v + v\rho^x((\sum_{i=1}^{d} w_i)^2 - \sum_{i=1}^{d} w_i^2)/\sum_{i=1}^{d} w_i^2}{\frac{1}{p}(c^2+v) - c^2 + v\rho^x((\sum_{i=1}^{d} w_i)^2 - \sum_{i=1}^{d} w_i^2)/\sum_{i=1}^{d} w_i^2} \\ &= \frac{v + v\rho^x(d(\cos\theta)^2 - 1)}{\frac{1}{p}(c^2+v) - c^2 + v\rho^x(d(\cos\theta)^2 - 1)}, \end{aligned} \tag{12}$$

Table 1. Statistical means of $(\cos\theta)^2$ and $d(\cos\theta)^2$ over all the convolutional layers on four representative networks.

Networks	CIFAR10		CIFAR100	
	$(\cos\theta)^2$	$d(\cos\theta)^2$	$(\cos\theta)^2$	$d(\cos\theta)^2$
PreResNet	0.03546	2.91827	0.03169	2.59925
ResNeXt	0.02244	14.78266	0.02468	14.72835
WRN	0.02292	52.73550	0.02118	44.31261
DenseNet	0.01538	3.83390	0.01390	3.43325

where $(\cos\theta)^2$ comes from the expression:

$$\frac{(\sum_{i=1}^{d} w_i)^2}{d\cdot\sum_{i=1}^{d} w_i^2} = \left(\frac{\sum_{i=1}^{d} 1\cdot w_i}{\sqrt{\sum_{i=1}^{d} 1^2}\sqrt{\sum_{i=1}^{d} w_i^2}}\right)^2 = (\cos\theta)^2, \tag{13}$$

and θ denotes for the angle between vector $\mathbf{w}$ and vector $\underbrace{(1\dots1)}_{\mathrm{m}}$. To prove that $d(\cos\theta)^2$ scales approximately linear to d, we made rich calculations w.r.t the term $d(\cos\theta)^2$ and $(\cos\theta)^2$ on four modern architectures[3] trained on CIFAR10/100 datasets (Table 1 and Figure 3). Based on Table 1 and Figure 3, we observe that $(\cos\theta)^2$ lies in $(0.01, 0.10)$ stably in every network and various datasets whilst $d(\cos\theta)^2$ tends to increase in parallel when d grows. From Equation (12), the inequation $Var^{Test}(X) \neq Var^{Train}(X)$ holds when $p < 1$. If we want $Var^{Test}(X)$ to approach $Var^{Train}(X)$, we need this term

$$\begin{aligned} \triangle(p, d) = \frac{Var^{Test}(X)}{Var^{Train}(X)} &= \frac{v\rho^x(d(\cos\theta)^2 - 1) + v}{v\rho^x(d(\cos\theta)^2 - 1) + \frac{1}{p}(c^2+v) - c^2} \\ &= \frac{v\rho^x + \frac{v(1-\rho^x)}{d(\cos\theta)^2}}{v\rho^x + \frac{(\frac{1}{p}-1)c^2 + v(\frac{1}{p}-\rho^x)}{d(\cos\theta)^2}} \end{aligned} \tag{14}$$

to approach 1. There are *two ways* to achieve $\triangle(p, d) \to 1$:

- $p \to 1$: maximizing the Dropout retain ratio p (ideally up to 1 which means Dropout is totally eliminated);
- $d \to \infty$: growing the width of channel exactly as the Wide ResNet did to enlarge d.

[3]For the convolutional filters which have larger than 1 filter size as $k \times k, k > 1$, we vectorise them by expanding its channel width d to $d \times k \times k$ while maintaining all the weights.

4. Statistical Experiments

We conduct extensive statistical experiments to check the correctness of above deduction in this section. Four modern architectures including DenseNet (Huang et al., 2016), PreResNet (He et al., 2016b), ResNeXt (Xie et al., 2017) and Wide ResNet (WRN) (Zagoruyko & Komodakis, 2016) are adopted on the CIFAR10 and CIFAR100 datasets.

Datasets. The two CIFAR datasets (Krizhevsky & Hinton, 2009) consist of colored natural scence images, with 32×32 pixel each. The train and test sets contain 50k images and 10k images respectively. CIFAR10 (C10) has 10 classes and CIFAR100 (C100) has 100. For data preprocessing, we normalize the data by using the channel means and standard deviations. For data augmentation, we adopt a standard scheme that is widely used in (He et al., 2016b; Huang et al., 2016; Larsson et al., 2016; Lin et al., 2013; Lee et al., 2015; Springenberg et al., 2014; Srivastava et al., 2015): the images are first zero-padded with 4 pixels on each side, then a 32×32 crop is randomly sampled from the padded images and at least half of the images are horizontally flipped.

Networks with Dropout. The four modern architectures are all chosen from the open-source codes[4] written in pytorch that can reproduce the results reported in previous papers. The details of the networks are listed in Table 2:

Table 2. Details of four modern networks in experiments. #P denotes for the amount of model parameters.

Model	#P on C10	#P on C100
PreResNet-110	1.70 M	1.77 M
ResNeXt-29, 8 × 64	34.43 M	34.52 M
WRN-28-10	36.48 M	36.54 M
DenseNet-BC (L=100, k=12)	0.77 M	0.80 M

Since the BN layers are already developed as the indispensible components of their body structures, we arrange Dropout that follows the two cases in Figure 2:

(a) We assign all the Dropout layers only and right before all the bottlenecks' last BN layers in these four networks, neglecting their possible Dropout implementations (as in DenseNet (Huang et al., 2016) and Wide ResNet (Zagoruyko & Komodakis, 2016)). We denote this design to be models of **Dropout-(a)**.

(b) We follow the assignment of Dropout in Wide ResNet (Zagoruyko & Komodakis, 2016), which finally improves WRNs' overall performances, to place the Dropout before the last Convolutional layer in every bottleneck block of PreResNet, ResNeXt and DenseNet. This scheme is denoted as **Dropout-(b)** models.

[4]Our implementations basicly follow the public code in https://github.com/bearpaw/pytorch-classification. The training details can also be found there. Our code for the following experiments would be released soon.

Statistics of variance shift. Assume a network $\mathcal{G}$ contains n BN layers in total. We arrange these BN layers from shallow to deep by giving them indices that goes from 1 to n accordingly. The whole statistical manipulation is conducted by following three steps:

(1) Calculate *moving_var*$_i, i \in [1, n]$: when $\mathcal{G}$ is trained until convergence, each BN layer obtains the moving average of neural variance (the unbiased variance estimate) from the feature-map that it receives during the entire learning procedure. We denote that variance to be *moving_var*. Since the *moving_var* for every BN layer is a vector (whose length is equal to the amount of channels of previous feature-map), we leverage its mean value to represent *moving_var* instead, for a better visualization. Further, we denote *moving_var*$_i$ as the *moving_var* of i-th BN layer.

(2) Calculate *real_var*$_i, i \in [1, n]$: after training, we fix all the parameters of $\mathcal{G}$ and set its state to be the evaluation mode (hence the Dropout would apply its inference policy and BN would freeze its moving averages of means and variances). The training data is again utilized for going through $\mathcal{G}$ within a certain of epochs, in order to get the real expectation of neural variances on the feature-maps before each BN layer. Data augmentation is also kept to ensure that every possible detail for calculating neural variances remains exactly the same with training. Importantly, we adopt the same moving average algorithm to accumulate the unbiased variance estimates. Similarly in (1), we let the mean value of real variance vector be *real_var*$_i$ before the i-th BN layer.

(3) Obtain *max*$(\frac{\textit{real\_var}_i}{\textit{moving\_var}_i}, \frac{\textit{moving\_var}_i}{\textit{real\_var}_i}), i \in [1, n]$: since we focus on the shift, the scalings are all kept above 1 by their reciprocals if possible in purpose of a better view. Various Dropout drop ratios $[0.0, 0.1, 0.3, 0.5, 0.7]$ are applied for clearer comparisons in Figure 4. The corresponding error rates are also included in each column.

Agreements between analyses and experiments about the relation between performance and variance shift. In these four columns of Figure 4, we discover that when the drop ratio is relatively small (i.e., 0.1), the green shift curves are all near the blue ones (i.e. models without Dropout), thus their performances are as well very close to the baselines. It agrees with our previous deduction that whenever in (a) or (b) case, decreasing drop ratio $1-p$ would alleviate the variance shift risks. Furthermore, in Dropout-(b) models (i.e., the last two columns) we find that, for WRNs, the curves with drop ratio $0.1, 0.3$ even 0.5 approaches closer to the one with 0.0 than other networks, and they all outperform the baselines. It also aligns with our analyses since WRN has a significantly larger channel dimension d, and it ensures that a slightly larger p would not explode the neural variance but bring the original benefits, which Dropout carries, back to the BN-equipped networks.

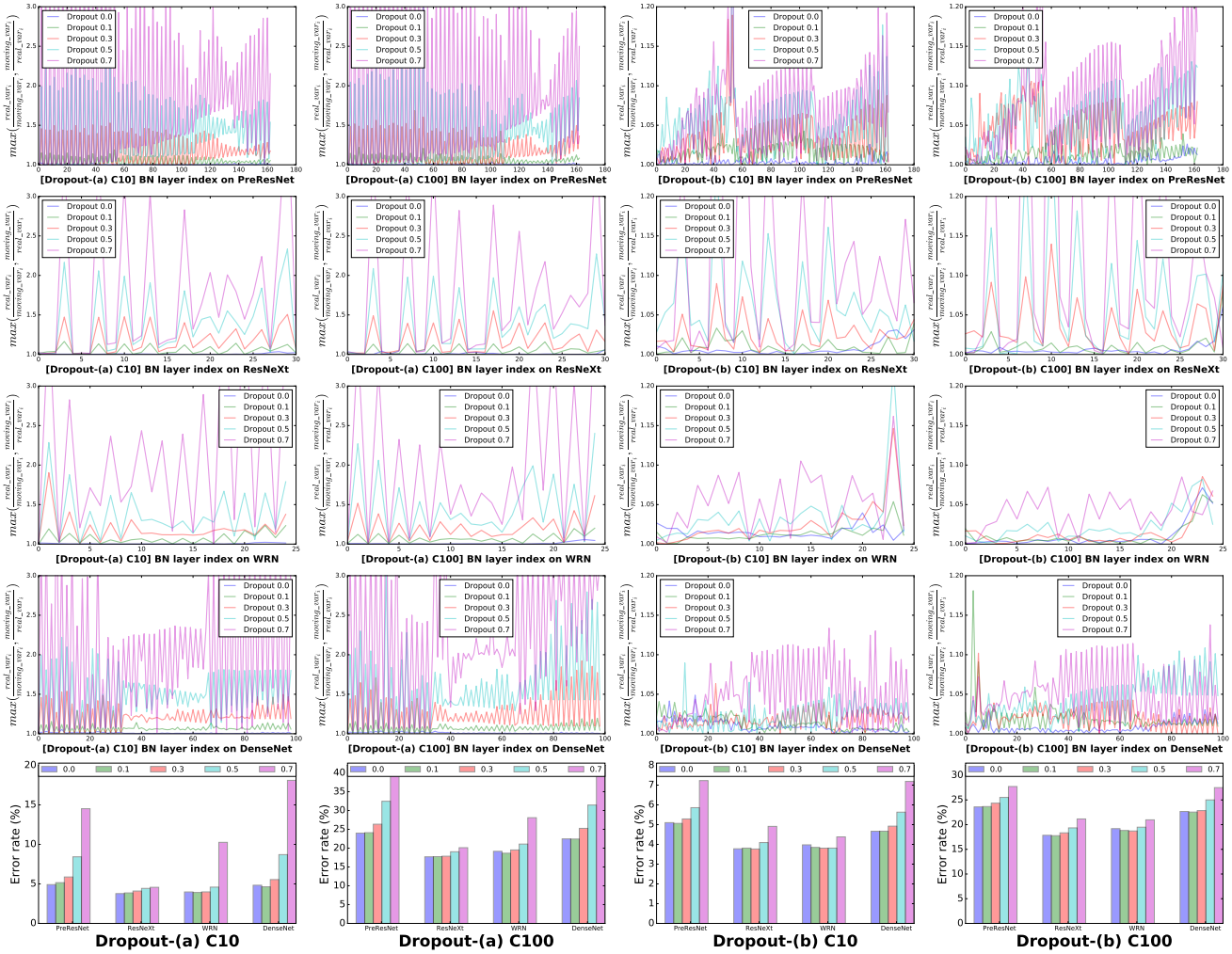

Figure 4. **See by columns.** Statistical visualizations about "variance shift" on BN layers of four modern networks w.r.t: 1) Dropout type; 2) Dropout drop ratio; 3) dataset, along with their test error rates (the fifth row). Obviously, WRN is less influenced by Dropout (i.e., small variance shift) when the Dropout-(b) drop ratio ≤ 0.5, thus it even enjoys an improvement with Dropout applied before BN.

Even the training data performs inconsistently between train and test mode. In addition, we also observe that for DenseNet and PreResNet (their channel d is relatively small), when their state is changed from train to test, even the training data cannot be kept with a coherent accuracy at last. In inference, the variance shift happens and it leads to an avalanche effect on the numerical explosion and instability in networks that finally changes the final prediction. Here we take the two models with drop ratio being 0.5 as an example, hence demonstrate that a large amount of training data would be classified inconsistently between train and test mode, despite their same model parameters (Figure 5).

Neural responses (of last layer before softmax) for training data are unstable from train to test. To get a clearer understanding of the numerical disturbance that the variance shift brings finally, a bundle of images (from training data) are drawn with their neural responses before the softmax layer in both train stage and test stage (Figure 6). From those pictures and their responses, we can find that with all the weights of networks fixed, only a mode transfer (from train to test) would change the distribution of the final responses even in the train set, and it leads to a wrong classification consequently. It proves that the predictions of training data differs between train stage and test stage when a network is equipped with Dropout layers before BN layers. Therefore, we confirm that the unstable numerical behaviors are the fundamental reasons for the performance drop.

Only an adjustment for moving means and variances would bring an improvement, despite all other parameters fixed. Given that the moving means and variances of BN would not match the real ones during test, we attempt to adjust these values by passing the training data again under the evaluation mode. In this way, the moving average algorithm (Ioffe & Szegedy, 2015) can also be applied. After shifting the moving statistics to the real ones by using the training data, we can have the model performed on the

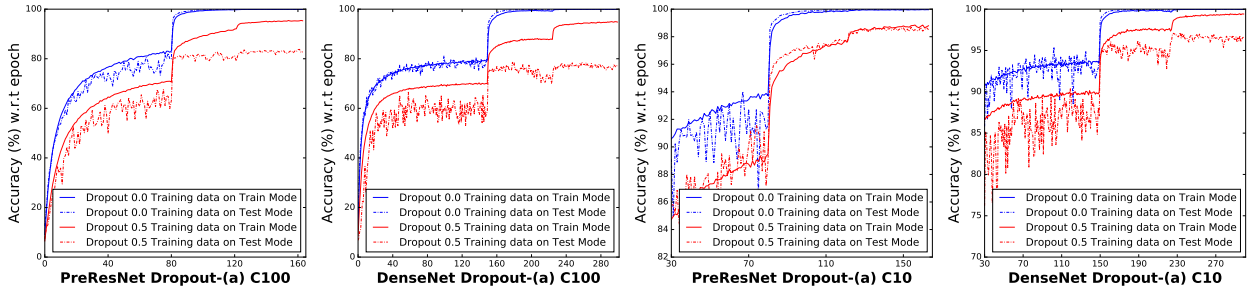

Figure 5. Accuracy by train epochs. Curves in blue means the train of these two networks without Dropout. Curves in red denotes the Dropout version of the corresponding models. These accuracies are all calculated from the training data, while the solid curve is under train mode and the dashed one is under evaluation mode. We observe the significant accuracy shift when a network with Dropout ratio 0.5 changes its state from train to test stage, with all network parameters fixed but the test policies of Dropout and BN applied.

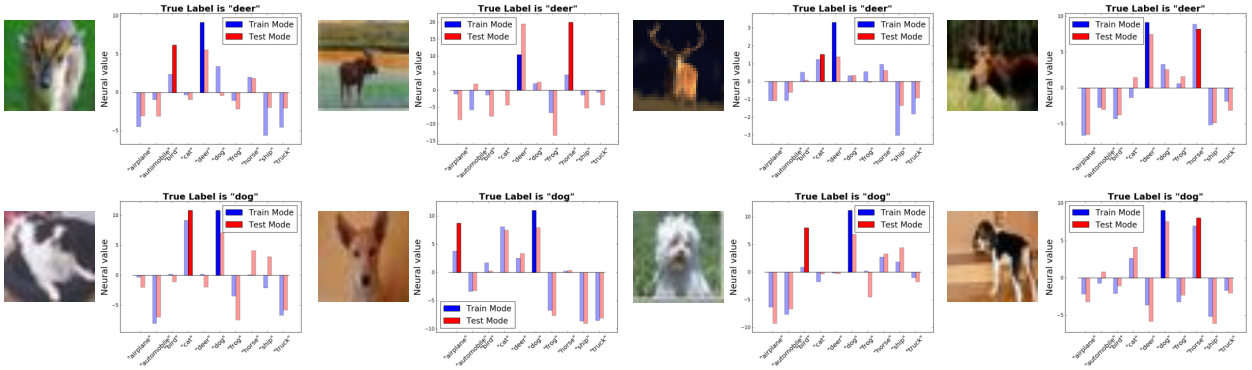

Figure 6. Examples of inconsistent neural responses between train mode and test mode of DenseNet Dropout-(a) 0.5 trained on CIFAR10 dataset. These samples are from the training data, whilst they are correctly classified by the model during learning yet erroneously judged in inference, despite all the fixed model parameters. Variance shift finally leads to the prediction shift that drops the performance.

Table 3. Adjust BN's moving mean/variance by running moving average algorithm on training data under test mode. These numbers are all averaged from 5 parallel runnings with different random initial seeds.

C10	Dropout-(a)		Dropout-(b)	
	0.5	0.5-Adjust	0.5	0.5-Adjust
PreResNet	8.42	**6.42**	5.85	**5.77**
ResNeXt	4.43	**3.96**	4.09	**3.93**
WRN	4.59	**4.20**	3.81	**3.71**
DenseNet	8.70	**6.82**	5.63	**5.29**
C100	Dropout-(a)		Dropout-(b)	
	0.5	0.5-Adjust	0.5	0.5-Adjust
PreResNet	32.45	**26.57**	25.50	**25.20**
ResNeXt	19.04	**18.24**	19.33	**19.09**
WRN	21.08	**20.70**	19.48	**19.15**
DenseNet	31.45	**26.98**	25.00	**23.92**

test set. From Table 3, All the Dropout-(a)/(b) 0.5 models outperform their baselines by having their moving statistics adjusted. Significant improvements (e.g., ~ 2 and ~ 4.5 gains for DenseNet on CIFAR10 and on CIFAR100 respectively) can be observed in Dropout-(a) models. It again verifies that the drop of performance could be attributed to the "variance shift": a more proper popular statistics with *smaller* variance shift could recall a bundle of erroneously classified samples back to right ones.

Table 4. Error rates after applying Dropout after all BN layers. These numbers are all averaged from 5 parallel runnings with different random initial seeds.

C10 drop ratio	0.0	0.1	0.2	0.3	0.5
PreResNet	5.02	4.96	5.01	**4.94**	5.03
ResNeXt	3.77	3.89	**3.69**	3.78	3.78
WRN	3.97	3.90	4.00	3.93	**3.84**
DenseNet	4.72	**4.67**	4.73	4.75	4.87
C100 drop ratio	0.0	0.1	0.2	0.3	0.5
PreResNet	23.73	**23.43**	23.65	23.45	23.76
ResNeXt	17.78	**17.77**	17.99	17.97	18.26
WRN	**19.17**	**19.17**	19.23	19.19	19.25
DenseNet	22.58	**21.86**	22.41	22.41	23.49

5. Strategies to Combine Them Together

Since we get a clear knowledge about the disharmony between Dropout and BN, we can easily develop several approaches to combine them together, to see whether an extra improvement could be obtained. In this section, we introduce two possible solutions in modifying Dropout. One is to avoid the scaling on feature-map before every BN layer, by only applying Dropout after the last BN block. Another is to slightly modify the formula of Dropout and make it less sensitive to variance, which can alleviate the shift problem and stabilize the numerical behaviors.

Table 5. Error rates after applying Dropout after all BN layers on the representative state-of-the-art models on ImageNet. These numbers are averaged from 5 parallel runnings with different random initial seeds. Consistent improvements can be observed.

ImageNet drop ratio	top-1		top-5	
	0.0	0.2	0.0	0.2
ResNet-200 (He et al., 2016b)	21.70	**21.48**	5.80	**5.55**
ResNeXt-101(Xie et al., 2017)	20.40	**20.17**	5.30	**5.12**
SENet (Hu et al., 2017)	18.89	**18.68**	4.66	**4.47**

Apply Dropout after all BN layers. According to above analyses, the variance shift only happens when there exists a Dropout layer before a BN layer. Therefore, the most direct and concise way to tackle this is to assign Dropout in the position where the subsequent layers donot include BN. Inspired by early works that applied Dropout on the fully connected layers in (Krizhevsky et al., 2012), we add only one Dropout layer right before the *softmax* layer in these four architectures. Table 4 indicates that such a simple operation could bring 0.1 improvements on CIFAR10 and reach up to 0.7 gain on CIFAR100 for DenseNet. Please note that the last-layer Dropout performs worse on CIFAR100 than on CIFAR10 generally since the training data of CIFAR100 is insufficient and these models may suffer from certain underfitting risks. We also find it interesting that WRN may not need to apply Dropout on each bottleneck block – only a last Dropout layer could bring enough or at least comparable benefits on CIFAR10. Additionally, we discover that in some previous work like (Hu et al., 2017), the authors already adopted the same tips in their winning solution on the ILSVRC 2017 Classification Competition. Since it didnot report the gain that last-layer Dropout brings, we made some additional experiments and evaluate several state-of-the-art models on the ImageNet (Russakovsky et al., 2015) validation set (Table 5) using a 224×224 centre crop evaluation on each image (where the shorter edge is first resized to 256). We observe consistent improvements when drop ratio 0.2 is employed after all BN layers on the large scale dataset.

Change Dropout into a more variance-stable form. The drawbacks of vanilla Dropout lie in the weight scale during the test phase, which may lead to a large disturbance on statistical variance. This clue could push us to think: if we find a scheme that functions like Dropout but carries a lighter variance shift, we may stabilize the numerical behaviors of neural networks, thus the final performance would probably enjoy a possible benefit. Here we take the **Figure 2 (a)** case as an example for investigation where the variance shift rate is $\frac{v}{\frac{1}{p}(c^2+v)-c^2} = p$ (we let $c = 0$ for simplicity). That is, if we set the drop ratio $(1-p)$ as 0.1, the variance would be scaled by 0.9 when the network is transferred from train to test. Inspired by the original Dropout paper (Srivastava et al., 2014) where the authors also proposed another form of Dropout that amounts to adding a Gaussian distributed random variable with zero mean and standard deviation equal to the activation of the unit, i.e., $x_i + x_i r$ and $r \sim \mathcal{N}(0,1)$, we modify r as a uniform distribution that lies in $[-\beta, \beta]$, where $0 \leq \beta \leq 1$. Therefore, each hidden activation would be $X = x_i + x_i r_i$ and $r_i \sim \mathcal{U}(-\beta, \beta)$. We name this form of Dropout as "Uout" for simplicity. With the mutually independent distribution between x_i and r_i being hold, we apply the form $X = x_i + x_i r_i$, $r_i \sim \mathcal{U}(-\beta, \beta)$ in train stage and $X = x_i$ in test mode. Similarly, in the simplified case of $c = 0$, we can deduce the variance shift again as follows:

$$\frac{Var^{Test}(X)}{Var^{Train}(X)} = \frac{Var(x_i)}{Var(x_i + x_i r_i)} = \frac{v}{E((x_i + x_i r_i)^2)} = \frac{v}{E(x_i^2) + 2E(x_i^2)E(r_i) + E(x_i^2)E(r_i^2)} = \frac{3}{3+\beta^2}. \tag{15}$$

Giving β as 0.1, the new variance shift rate would be $\frac{300}{301} \approx 0.9966777$ which is much closer to 1.0 than the previous 0.9 in **Figure 2 (a)**. A list of experiments is hence employed based on those four modern networks under Dropout-(b) settings w.r.t β (Table 6). We find that "Uout" would be less affected by the insufficient training data on CIFAR100 than applying the last-layer Dropout, which indicates a superior property of stability. Except for ResNeXt, nearly all the architectures achieved up to $0.2 \sim 0.3$ increase of accuracy on both CIFAR10 and CIFAR100 dataset.

Table 6. Apply new form of Dropout (i.e. Uout) in Dropout-(b) models. These numbers are all averaged from 5 parallel runnings with different random initial seeds.

C10 β	0.0	0.1	0.2	0.3	0.5
PreResNet	5.02	5.02	**4.85**	4.98	4.97
ResNeXt	3.77	3.84	3.83	**3.75**	3.79
WRN	3.97	3.96	**3.80**	3.90	3.84
DenseNet	4.72	4.70	4.64	4.68	**4.61**
C100 β	0.0	0.1	0.2	0.3	0.5
PreResNet	23.73	23.73	23.62	**23.53**	23.77
ResNeXt	17.78	**17.74**	17.77	17.83	17.86
WRN	19.17	19.07	18.98	18.95	**18.87**
DenseNet	22.58	22.39	22.57	22.35	**22.30**

6. Conclusion

In this paper, we investigate the "variance shift" phenomenon when Dropout layers are applied before Batch Normalization on modern neural networks. We discover that due to their distinct test policies, neural variance would be improper and shifted as the information flows in inference, and it leads to the unexpected final predictions that drops the performance. To avoid the variance shift risks, we next explore two strategies, and they are proved to work well in practice. We highly recommand that researchers could take these solutions to boost their models' performance if further improvement is desired, since their extra cost is nearly free and they are easy to be implemented.

References

Hannun, Awni, Case, Carl, Casper, Jared, Catanzaro, Bryan, Diamos, Greg, Elsen, Erich, Prenger, Ryan, Satheesh, Sanjeev, Sengupta, Shubho, Coates, Adam, et al. Deep speech: Scaling up end-to-end speech recognition. *arXiv preprint arXiv:1412.5567*, 2014.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Deep residual learning for image recognition. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 770–778, 2016a.

He, Kaiming, Zhang, Xiangyu, Ren, Shaoqing, and Sun, Jian. Identity mappings in deep residual networks. In *European Conference on Computer Vision*, pp. 630–645. Springer, 2016b.

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

Hu, Jie, Shen, Li, and Sun, Gang. Squeeze-and-excitation networks. *arXiv preprint arXiv:1709.01507*, 2017.

Huang, Gao, Liu, Zhuang, Weinberger, Kilian Q, and van der Maaten, Laurens. Densely connected convolutional networks. *arXiv preprint arXiv:1608.06993*, 2016.

Ioffe, Sergey and Szegedy, Christian. Batch normalization: Accelerating deep network training by reducing internal covariate shift. In *International Conference on Machine Learning*, pp. 448–456, 2015.

Kim, Yoon, Jernite, Yacine, Sontag, David, and Rush, Alexander M. Character-aware neural language models. In *the Association for the Advance of Artificial Intelligence*, pp. 2741–2749, 2016.

Krizhevsky, Alex and Hinton, Geoffrey. Learning multiple layers of features from tiny images. 2009.

Krizhevsky, Alex, Sutskever, Ilya, and Hinton, Geoffrey E. Imagenet classification with deep convolutional neural networks. In *Advances in Neural Information Processing Systems*, pp. 1097–1105, 2012.

Larsson, Gustav, Maire, Michael, and Shakhnarovich, Gregory. Fractalnet: Ultra-deep neural networks without residuals. *arXiv preprint arXiv:1605.07648*, 2016.

Lee, Chen-Yu, Xie, Saining, Gallagher, Patrick, Zhang, Zhengyou, and Tu, Zhuowen. Deeply-supervised nets. In *Proceedings of the International Conference on Artificial Intelligence and Statistics*, pp. 562–570, 2015.

Lin, Min, Chen, Qiang, and Yan, Shuicheng. Network in network. *arXiv preprint arXiv:1312.4400*, 2013.

Russakovsky, Olga, Deng, Jia, Su, Hao, Krause, Jonathan, Satheesh, Sanjeev, Ma, Sean, Huang, Zhiheng, Karpathy, Andrej, Khosla, Aditya, Bernstein, Michael, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.

Springenberg, Jost Tobias, Dosovitskiy, Alexey, Brox, Thomas, and Riedmiller, Martin. Striving for simplicity: The all convolutional net. *arXiv preprint arXiv:1412.6806*, 2014.

Srivastava, Nitish, Hinton, Geoffrey E, Krizhevsky, Alex, Sutskever, Ilya, and Salakhutdinov, Ruslan. Dropout: a simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(1):1929–1958, 2014.

Srivastava, Rupesh K, Greff, Klaus, and Schmidhuber, Jürgen. Training very deep networks. In *Advances in Neural Information Processing Systems*, pp. 2377–2385, 2015.

Szegedy, Christian, Liu, Wei, Jia, Yangqing, Sermanet, Pierre, Reed, Scott, Anguelov, Dragomir, Erhan, Dumitru, Vanhoucke, Vincent, and Rabinovich, Andrew. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 1–9, 2015.

Szegedy, Christian, Vanhoucke, Vincent, Ioffe, Sergey, Shlens, Jon, and Wojna, Zbigniew. Rethinking the inception architecture for computer vision. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pp. 2818–2826, 2016.

Szegedy, Christian, Ioffe, Sergey, Vanhoucke, Vincent, and Alemi, Alexander A. Inception-v4, inception-resnet and the impact of residual connections on learning. In *the Association for the Advance of Artificial Intelligence*, pp. 4278–4284, 2017.

Xie, Saining, Girshick, Ross, Dollár, Piotr, Tu, Zhuowen, and He, Kaiming. Aggregated residual transformations for deep neural networks. In *IEEE Conference on Computer Vision and Pattern Recognition*, pp. 5987–5995. IEEE, 2017.

Zagoruyko, Sergey and Komodakis, Nikos. Wide residual networks. *arXiv preprint arXiv:1605.07146*, 2016.

Zhang, Xiangyu, Zhou, Xinyu, Lin, Mengxiao, and Sun, Jian. Shufflenet: An extremely efficient convolutional neural network for mobile devices. *arXiv preprint arXiv:1707.01083*, 2017.