

Inner Ensemble Nets

Abduallah Mohamed¹

Muhammed Mohaimin Sadiq¹

Ehab AlBadawy²

Mohamed Elhoseiny^{3,**}

Christian Claudel^{1,**}

¹UT Austin

²UAlbany

³KAUST, Stanford

^{**} Equal Advising

Abstract

We introduce Inner Ensemble Networks (IENs) which reduce the variance within the neural network itself without an increase in the model complexity. IENs utilize ensemble parameters during the training phase to reduce the network variance. While in the testing phase, these parameters are removed without a change in the enhanced performance. IENs reduce the variance of an ordinary deep model by a factor of $1/m^{L-1}$, where m is the number of inner ensembles and L is the depth of the model. Also, we show empirically and theoretically that IENs lead to a greater variance reduction in comparison with other similar approaches such as dropout and maxout. Our results show a decrease of error rates between 1.7% and 17.3% in comparison with an ordinary deep model. Code is available at https://github.com/abduallahmohamed/inner_ensemble_nets.git.

1 Introduction

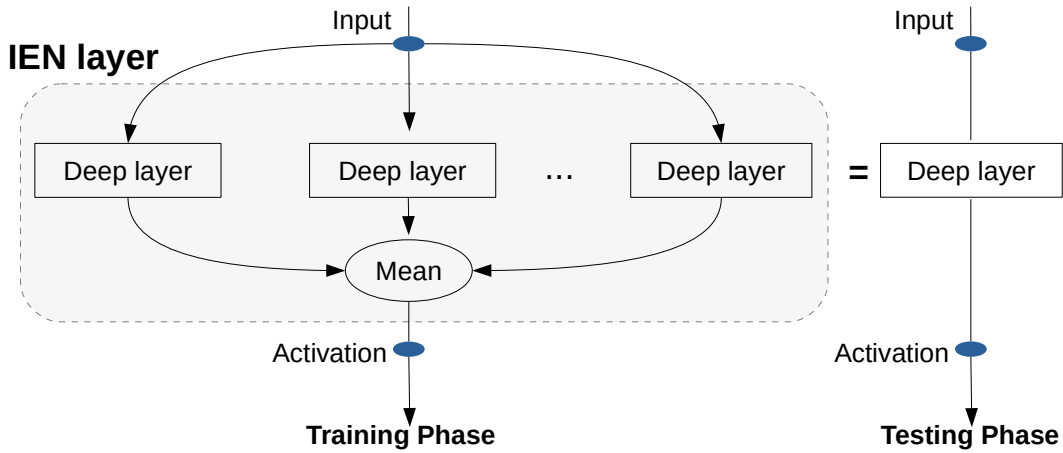


Figure 1: Basic IEN concept. During the training phase, inner ensemble weights are used to achieve better performance. In the testing phase, these extra weights are no longer necessary. The final model parameters count is the same as that of the original model without additional ensemble parameters.

Ensemble learning [32, 43] is based on a combination of multiple models predictions trained over the same dataset or a random subsets of the dataset to improve model performance. Ensemble methods have been widely used in deep learning [30, 6, 10, 41] to improve overall model accuracy. Combining neural networks in ensembles is known to reduce the variance in the prediction. In other

words, the ensemble of networks generalizes better to unseen data than a single network [22, 13, 44]. However, there is a concern regarding the number of parameters of the final model, which involves the parameters of each constituent model. The large number of parameters in the final model decreases inference speed and increases model storage size, which can be problematic for some applications. In this paper, our objective is to find a solution that leads to an overall reduction in variance resulting from the ensembles, yet preserves the number of parameters of the underlying model.

We introduce Inner Ensemble Networks (IENs) to overcome the problem of the increased number of parameters in regular ensembles. IEN uses m instances of the same layer and applies an average operation on the output of these layers as illustrated in Figure 1. IEN can be applied to each layer of a selected base model architecture. We hypothesize that the overall variance of the model can be decomposed into sub-variances within the model layers itself. By using IENs, we reduce the sub-variances of each layer resulting in an overall variance reduction of the model. Therefore, we primarily explore the averaging method in our setting. Now we refer to regular ensembles as outer ensembles. The performance of IENs is between that of an ordinary deep model and an outer ensemble of multiple instances of the same model.

In this work, we discuss the theoretical aspects of variance reduction resulting from applying IENs. We discuss how to remove the excess weights used during the training phase theoretically and we verify this empirically. We also contrast IEN variance reduction with closely related approaches such as dropout [37] and maxout [15]. When using IEN during the training phase, m times the number of parameters in the base model are required to do an inner ensemble. This could be expensive if applied to Fully Connected (FC) layers. We show empirically that using IEN on Convolutional Neural Network (CNN) layers alone results in a performance that exceeds that of a base model. This makes the training cheaper because of the shared parameters property of CNNs. We also show that using IEN on both CNNs and Fully Connected (FC) layers similarly results in a performance which exceeds that of base models, but at the cost of increased training time. Lastly, we extend IEN as a Neural Architecture Search (NAS) [45] cell. We applied NAS searching algorithms to discover new cells. The discovered cells show a preference for using IEN as a cell operator.

2 Related work

Ensemble methods are meta-algorithms that combine a set of independently trained networks into one predictive model in order to improve performance. Other methods include bagging and boosting which reduce predictive variance and bias [25]. Averaging the results of several models is one of the most common approaches and is extensively adopted in modern deep learning practice [35, 31]. One variation is to create ensembles of the same model at different training epochs [30]. [3] proposed using inner ensembles to make learning decisions within the learning algorithms of classic machine learning techniques. Other approaches such as [4] used a model compression technique, in which they trained a shallow network using an ensemble of networks. Yet, it has a high cost of first training the ensembles and then the actual shallow model unlike IENs which train a single model. [26] developed a method that averages the output layer of a model by encouraging diversity in subsets of the output layer neurons. A main concern is the complexity of the method which is not a simple plug in method unlike IENs. In IENs you simply replace the CNN or FC layers with IEN layers without the need to change anything else. Another concern in [26] is that it forces a specific loss function unlike IENs which is agnostic from the choice of a loss function. Though inspired by ensemble methods that average independent networks, the proposed IEN structure is fundamentally different in that a) it only needs the extra ensemble weights during the training phase and b) it trains the ensembles jointly in a single model.

We could state that the idea of IEN existed implicitly in previous methods, starting from maxout [15]. Maxout replicates deep layers and takes the maximum of the response coming from these replicas. Maxout can be considered as an ensemble that uses max instead of average. Yet, we show both theoretically and empirically that maxout is not a desirable option and IEN is a better alternative. Dropout [37, 5] can be seen as a geometric mean of several sub-networks within a deep model. We believe that IEN aligns with dropout as they are both only applied during the training phase. However, IEN can be applied on every layer of the deep model unlike dropout. Dropout was originally designed to be applied on FC layers. When it comes to CNNs, dropout is applied carefully [27] and cannot be utilized to the fullest extent. We also show theoretically IEN variance reduction is greater than that of dropout when $m \geq 2$.

Also, the concept of IEN implicitly existed in previous deep model designs, starting from ResNeXt [42]. ResNeXt widened each CNN layer by introducing complicated pathways within ResNet [17] blocks. However, they used an activation function within these layers and added these paths into a single output node. Another model design was GoogLeNet [38], where the idea of multiple paths was introduced, but using different kernel sizes unlike ResNeXt. Both models had the concept of parallel layers where their results were aggregated at the end. Though IEN is similar to these models, it is different in the core design by virtue of duplicating the same model layers and averaging the results before the activation function.

3 IEN theoretical analysis

In this section, we start with analyzing IEN variance response. We discuss the method of removing the ensemble weights. Then we contrast maxout and dropout variance response with that of IEN.

3.1 IEN variance response

In this section we show that IEN ensembles of size $m \in \mathbb{Z}^{>1}$ decrease the overall variance of a deep net by a factor of:

$$\frac{1}{m^{L-1}}, \text{ Where } L \text{ is the number of layers in the network.} \quad (1)$$

We end up with a new initialization that prevents the explosion or degradation of the gradients. We follow the leads of [18] and [14] to derive the variance analysis for IEN. Let us assume a deep model with L layers. The l th layer has a response $\mathbf{y}_l$ from an input $\mathbf{x}_l$ multiplied by a learnable weight matrix $\mathbf{w}_l$, where $\mathbf{y}_l$, $\mathbf{x}_l$ and $\mathbf{w}_l$ are all tensors without a bias term. This can be formulated as:

$$\mathbf{y}_l = \mathbf{w}_l \mathbf{x}_l \quad (2)$$

An IEN layer can be defined as:

$$\mathbf{y}_l^{\text{IEN}} = \frac{1}{m} \sum_{i \in m} \mathbf{w}_l^i \mathbf{x}_l \quad (3)$$

We make the following assumptions:

- $\mathbf{w}_l^i = \{w_{kj}^i : k \in n, j \in d\} \in \mathbb{R}^{n \times d}$ elements are initialized to be mutually independent with zero mean and a symmetric distribution sharing the same distribution variance.
- $\mathbf{x}_l = \{x_i : i \in d\} \in \mathbb{R}^{d \times 1}$ elements are initialized to be mutually independent sharing the same distribution with zero mean.
- $\mathbf{w}_l$ and $\mathbf{x}_l$ are independent from each other.

Noting that $\mathbf{y}_l^{\text{IEN}} = \{y_i : i \in n\} \in \mathbb{R}^{n \times 1}$. The variance of each $y_l^{\text{IEN}} \in \mathbf{y}_l^{\text{IEN}}$ is:

$$\text{Var}[y_l^{\text{IEN}}] = n_l \text{Var}\left[\frac{1}{m} \sum_{i \in m} w_l^i x_l\right] = n_l \frac{1}{m^2} \text{Var}\left[\sum_{i \in m} w_l^i x_l\right] \quad (4)$$

Because w_l^i and x_l are independent, which implies they are uncorrelated, we can state:

$$\text{Var}[y_l^{\text{IEN}}] = n_l \frac{1}{m^2} \sum_{i \in m} \text{Var}[w_l^i x_l] = n_l \frac{1}{m^2} \text{Var}[x_l] \sum_{i \in m} \text{Var}[w_l^i] \quad (5)$$

Because all elements of w_l^i share the same distribution parameters, their variances are equal. We can define $\text{Var}[w_l^i] = \text{Var}[w_l]$ and by their independence:

$$\text{Var}[y_l^{\text{IEN}}] = n_l \frac{1}{m} \text{Var}[x_l] \text{Var}[w_l] \quad (6)$$

By using the the results and assumptions made at Appendix A.2 given that $x_l = f(y_{l-1})$, where f is an activation function and β^2 is the gain of f , we have:

$$\text{Var}[y_l^{\text{IEN}}] = n_l \beta^2 \frac{1}{m} \text{Var}[w_l] \text{Var}[y_{l-1}] \quad (7)$$

For cascaded l layers and with the fact that $\text{Var}[y_1]$ represents the variance of the input layer, we arrive at:

$$\text{Var}[y_l^{\text{IEN}}] = \text{Var}[y_1] \left(\prod_{l=2}^L \beta^2 n_l \frac{1}{m} \text{Var}[w_l] \right) \quad (8)$$

An ordinary deep model variance response [18] is:

$$\text{Var}[y_L] = \text{Var}[y_1] \left(\prod_{l=2}^L \beta^2 n_l \text{Var}[w_l] \right) \quad (9)$$

From equation 8 and 9 the variance of response using IEN is $\frac{1}{m^{L-1}}$ times less than using an ordinary deep model. This states that going deeper using IEN layers or wider using m will lead to better generalization of the model. We verified this empirically in the experiments section.

We also want the multiplication in 8 to have a proper scalar in order to avoid reduction or magnification of the input signal, we want:

$$\frac{\beta^2 n_l}{m} \text{Var}[w_l] = 1, \forall l \quad (10)$$

So we need to initialize our weights to be: $w_l^i \sim \mathcal{N}(0, \frac{m}{\beta^2 n})$. For simplicity, we use the same initialization for the first layer.

3.2 Reverting back to the original network size

Though the IEN reduces the variance of the network, we still have the burden of the extra parameters. Because of the training mechanism of IEN, all m ensembles receive the same error signal. This results in weights that are close to each other in value, but not exactly the same. Because these weights are similar, one might think that the best approach is to average these weights. Yet, averaging will treat all of them equally and this is not fair. Our goal is to revert back to the original weights after training without a loss of accuracy. We utilize the concept of inverse variance weighting [16], which is quite applicable in our case and accounts for the slight variation in the weights.

Define $\text{Var}[\{w : w \in \mathbf{w}_l^i\}] = \sigma_l^{2,i}$. We define the inverse variance weight α to be $\alpha_l^i = \frac{1/\sigma_l^{2,i}}{\sum_{i \in m} 1/\sigma_l^{2,i}}$.

Then the final weight of IEN $\tilde{\mathbf{w}}_l$ from using inverse variance weighting becomes:

$$\tilde{\mathbf{w}}_l = \sum_{i \in m} \alpha_l^i \mathbf{w}_l^i \quad (11)$$

Then, our IEN layer becomes: $\mathbf{y}_l^{\text{IEN}} = \tilde{\mathbf{w}}_l \mathbf{x}_l$ which has the same number of parameters as before using the IEN, but with the added benefit of variance reduction. Hence, we were able to revert back to the original model size for the inference stage, yet we utilized the extra parameters of IEN during the training phase. Natural questions arise regarding whether the method of inverse variance weighting is applicable to maxout and whether it is applicable to m separately trained models. The answer to both is no. We verified these findings empirically in the experiments section.

3.3 Connection with dropout

Dropout [37] is considered as a geometric mean of several small networks [5]. Here we try to connect dropout with IEN and contrast the variance performance. We want to note that we do not introduce IEN as an alternative for dropout. It is simply a tool to be used alongside other deep learning regularization tools. Dropout can be defined as:

$$y_l^{\text{dropout}} = \delta_l w_l x_l \quad (12)$$

Where $\delta_l \sim \text{Bernoulli}(p)$ of probability p . From Appendix B, the variance of a dropout layers is upper bounded as follows:

$$\text{Var}[y_L^{\text{dropout}}] \leq \text{Var}[y_1] \left(\prod_{l=2}^L \beta^2 n_l \frac{1}{2} \text{Var}[w_l] \right) \quad (13)$$

Thus, dropout decreases the variance response at most:

$$\leq \frac{1}{2^{L-1}} \text{ times less than using an ordinary deep model.} \quad (14)$$

Comparing equations 14 and 1, we conclude that IEN outperforms dropout if used alone when $m > 2$.

3.4 Connection with maxout

Maxout [15] is a universal approximator. It shares a common setting with IEN in terms of having multiple replicas of the weights and it selects the max response out of these replicas. One disadvantage of maxout is that the required number of parameters in both training and inference stays the same, unlike IEN with inverse variance weighting that solves the problem for inference as stated in section 3.2. A maxout layer is defined as:

$$\mathbf{y}_l^{\text{maxout}} = \max\{\mathbf{w}_l^i \mathbf{x}_l\}_i^m \quad (15)$$

Because the usage of max makes the analysis of variance much harder, we provide three views for the variance of maxout. The first one is a general upper bound (appendix C) on the variance of maxout, which is not tight, but is helpful in our analysis. The bound is:

$$\text{Var}[y_L^{\text{maxout}}] \leq \text{Var}[y_1] \left(\prod_{l=2}^L n_l m \beta^2 \text{Var}[w_l] \right) \quad (16)$$

which suggests that maxout might result in an explosion in the variance by a factor of

$$m^{L-1} \text{ more than using an ordinary deep model.} \quad (17)$$

in comparison with both IEN and even an ordinary deep model. We observe this behavior in Table 1.

The second bound of maxout states the lowest possible variance [7] (derived in appendix D) it can achieve under an assumption that $(w_l^i x_l) \sim \mathcal{N}(0, 1)$ and a linear activation function is:

$$\text{Var}[y_l^{\text{maxout}}] \geq n_l \frac{c}{\log m}, c > 0 \quad (18)$$

The IEN variance reduction gain under the later assumptions and by using equation 1 is:

$$\text{Var}[y_l^{\text{IEN}}] = n_l \frac{1}{m} \quad (19)$$

This maxout bound shows that as the number of ensembles m increases, the variance decreases, controlled by the value of c . The issue is that this bound is only for a one layer deep model and it is difficult to extend it to multi-layer models. Yet it gives some insights. Depending on the value of c in equation 18, the IEN could perform better or worse. We highlight that these findings are limited and only valid for the aforementioned assumptions.

The last bound is the asymptotic bound by using extreme value theorem [23] (derived in appendix E). When m is sufficiently large under the assumptions that $(w_l^i x_l) \sim \mathcal{N}(0, \sigma^2)$ and f is a linear activation function, the bound is:

$$\text{Var}[y_l^{\text{maxout}}] \approx n_l \frac{\pi^2}{6} \frac{\sigma^2}{2 \ln m} \quad (20)$$

suggesting that the asymptotic maxout bound is:

$$\approx \frac{\pi^2}{12 \ln(m)} \text{ times less than ordinary deep model} \quad (21)$$

3.5 Summary of the variance gains

Figure 2 summarizes the variance behaviour in IEN, dropout and maxout. We assume a one layer deep model to have common settings that aligns with all derived bounds and f is a linear activation function. We used equation 19 for IEN, and equation 14 for dropout and equations 18, 20 and 17 for maxout. We observe that IEN variance reduction surpasses all previous methods and the different maxout variance bounds except when c is extremely small in maxout lower variance bound. We also observe that the maxout upper bound explodes compared to IEN and dropout. Dropout variance reduction remains the same.

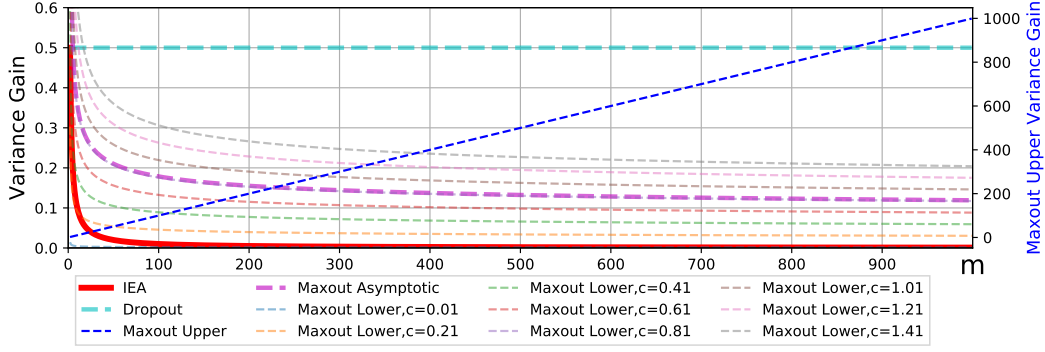


Figure 2: The variance gain of IEN, maxout and dropout versus the number of ensembles in a single linear layer deep model. Please note that maxout upper variance gain has a different scale.

4 Experiments & discussions

We wanted to verify that IEN generalized the deep model better than other approaches. For this we chose three well-known deep models based on the extent of their residual connections, starting from VGG [36] which has no residual connections, then ResNet [17] which has one residual connection, and finally DenseNet [19] which is a fully residual connected net. We believe that benchmarking against these models will give an overview that be can generalized to other architectures. We use both CIFAR-10 and CIFAR-100 [21] for all of our experiments as the former shows the performance on a small classification task and the latter on a larger classification task. We also choose to experiment on image classification models as they use both CNN and FC layers. The same hyper-parameters were applied across each set of model experiments.

Table 1: Error rate mean and std of IEN, maxout and the original model design on different deep model architectures. The lower, the better. The subscript $\tilde{w}$ indicates results using the weight downsizing method from section 3.2. $+FC$ stands for IEN applied on both FC and CNN layers. Maxout results are only on CNN layers. The blue colored models have the exact same number of parameters. The rest have the same number of parameters that scales with m . IEN and maxout have $m = 4$.

Dataset		ResNet56	ResNet110	DenseNet40-12	DenseNet100-12	VGG16	VGG19
CIFAR-10	IEN $\tilde{w}$	6.93$\pm$0.26	6.27$\pm$0.32	7.50 $\pm$ 0.18	5.22$\pm$0.19	6.80$\pm$0.18	6.79$\pm$0.30
	IEN	6.93$\pm$0.26	6.28$\pm$0.32	7.5 $\pm$ 0.18	5.22$\pm$0.19	6.79$\pm$0.19	6.79$\pm$0.31
	IEN+FC $\tilde{w}$	6.49$\pm$0.04	5.94$\pm$0.26	6.89$\pm$0.20	5.04$\pm$0.07	6.85 $\pm$ 0.18	7.15 $\pm$ 0.27
	Maxout $\tilde{w}$	89.99 $\pm$ 0	90 $\pm$ 0	89.70 $\pm$ 0.51	90 $\pm$ 0	90 $\pm$ 0	90 $\pm$ 0
	Maxout	7.98 $\pm$ 0.32	6.32 $\pm$ 0.12	7.34 $\pm$ 0.155	6.16 $\pm$ 0.54	7.71 $\pm$ 0.15	8.16 $\pm$ 0.18
	Base $\tilde{w}$	89.99	90	90	90	90	90
	Base	8.38 $\pm$ 1.2	6.38 $\pm$ 0.48	7.20 $\pm$ 0.15	5.6 $\pm$ 0.12	7.00 $\pm$ 0.08	7.02 $\pm$ 0.08
CIFAR-100	IEN $\tilde{w}$	29.34$\pm$0.49	28.17 $\pm$ 0.11	29.37$\pm$0.30	23.76$\pm$0.45	28.49$\pm$0.39	29.40$\pm$0.22
	IEN	29.34$\pm$0.49	28.16 $\pm$ 0.10	29.37$\pm$0.30	23.76$\pm$0.45	28.50$\pm$0.39	29.41$\pm$0.23
	IEN+FC $\tilde{w}$	28.20$\pm$0.19	27.34$\pm$0.28	29.76$\pm$0.19	23.67$\pm$0.37	29.33 $\pm$ 0.10	31.81 $\pm$ 0.19
	Maxout $\tilde{w}$	99.01 $\pm$ 0.01	99 $\pm$ 0	99 $\pm$ 0	99 $\pm$ 0	99 $\pm$ 0	99 $\pm$ 0
	Maxout	31.8 $\pm$ 1.69	29.47 $\pm$ 0.84	30.49 $\pm$ 0.75	28.88 $\pm$ 5.72	31.68 $\pm$ 0.65	34.32 $\pm$ 0.30
	Base $\tilde{w}$	98.96	99.14	99	90	99	99
	Base	29.97 $\pm$ 0.71	27.83 $\pm$ 0.64	29.85 $\pm$ 0.39	24.01 $\pm$ 0.14	29.27 $\pm$ 0.27	30.92 $\pm$ 0.49

4.1 IEN behavior analysis

Where to apply IEN: IEN can be applied to both CNN and FC layers. The question that arises concerns the cost of training time which is correlated with the number of parameters. Another question is whether it is better to apply IEN to just CNN layers or both CNN and FC layers. From Table 1, applying IEN to CNN layers only, IEN $\tilde{w}$, yields better results than the base model as observed for ResNet, VGG an DenseNet. Also, it might not be enough to apply it only to CNN layers, as observed in DenseNet40-12 results. Yet, when it is applied to FC layers, IEN+FC $\tilde{w}$, too it will yield better results than the base model. So, the model designer has a choice. If IEN is applied to CNN layers only, it will save training time and enhance the performance. If applied to both CNN and FC layers it will increase the training time, but will result in a better performance than the former.

Weight downsizing property: We denote the downsized model using the method from section 3.2 with subscript $\tilde{w}$. We observe from Table 1 that downsized IEN models which have the same number of parameters as the base model do have the same performance as the one with the full ensemble parameters. This property was verified across different architectures as shown in the same table. Also, it holds for applying IEN on CNN or FC layers. We also wanted to verify if this property can be extended to the base model or the maxout model. We applied $\tilde{w}$ on both of them and as we observe from Table 1 that their performance was extremely poor and close to random guessing. We observe that the weight downsizing property was only successful in IEN for two reasons. Firstly, the mean is a linear operator allowing the inner ensembles weights to receive the same error signal during the training phase. Secondly, the weights were initialized from the same distribution. This leads the weights to behave very similarly, resulting in closely related weights. This condition is key to having the inverse variance weighting work out properly.

Connection with the variance analysis: From Table 1, we observe that IEN outperforms previous methods such as maxout and the base model. We now recall from the variance reduction gain of IEN equation 1 that going deeper leads to better results. We notice this happens in both ResNet and DenseNet. Yet, it does not hold for VGG-16 and VGG-19 which is the same case for maxout and the base model. This suggests the necessity of residual connections as in [40]. Aside from VGG results, our variance analysis of IEN aligns with the empirical findings. For maxout, we notice in some cases such as ResNet56 results for CIFAR-10, that it performs better than the base model. This is related to the maxout lower bound that was discussed earlier. However, in some cases such as ResNet56 for CIFAR-100, maxout performs far worse than the base model. This aligns too with the upper and asymptotic variance bounds of maxout.

Effect of number of ensembles m : We also wanted to study the effect of number of ensembles m on accuracy of the models. Figure 3 illustrates this affect. We notice that IEN with $m = [2, 4, 8]$ outperforms maxout with the same m . Most of the time maxout performs worse than the base model. In a few cases such as CIFAR-10 ResNet56, maxout might perform better than the base model but not better than IEN. Most of the time IEN and IEN+FC outperform the base model. Lastly, using $m = [2, 4]$ is much better than using $m = 8$. This suggests that going higher with the number of inner ensembles is not desirable. This also aligns with the typical behavior of an outer ensemble [33].

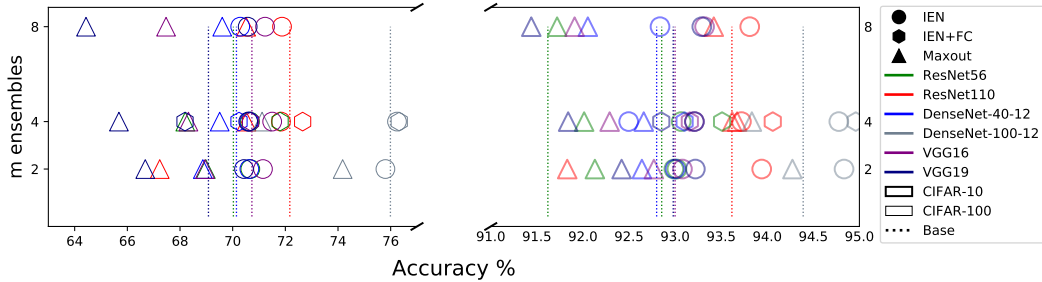


Figure 3: The effect of changing m versus model mean accuracy. IEN, maxout and base model accuracies are shown. CIFAR-10 results are the thin line shapes, while CIFAR-100 results are the thick line shapes. Colors represents different architectures. Dashed line is the base model mean accuracy.

4.2 Comparison with outer-ensemble

We denote regular ensembles as outer ensembles to distinguish them from inner ensembles. From Table 2, an outer ensemble of IENs exceeds in performance an outer ensemble of base models and maxout models using the same number of parameters except in a few cases. From Table 2 and Table 1 we notice that an outer ensemble of the base model outperforms single model IEN, but an IEN model is better than a single base model. This indicates that IEN performance stands in between a single regular model and an ensemble of the same model.

Table 2: Results of outer-ensemble of the models. Each model were trained 3 times and their predictions were averaged. The metric reported is the mean and std of the error rate. The lower the better. The subscript $\tilde{w}$ indicates results with the weight downsizing method from section 3.2. +FC stands for IEN applied on Fully Connected layer. All IENs and Maxouts have $m = 4$. The base model ensemble results are from 4 trained models.

Dataset		ResNet56	ResNet101	DenseNet40-12	DenseNet100-12	VGG16	VGG19
CIFAR-10	IEN $_{\tilde{w}}$	5.49	5.00	5.91	4.02	5.68	5.71
	IEN+FC $_{\tilde{w}}$	5.05	4.95	5.21	4.04	5.70	6.02
	Maxout	6.43	5.20	5.66	4.87	6.24	6.66
	Base	5.68	4.97	5.46	4.59	5.61	5.67
CIFAR-100	IEN $_{\tilde{w}}$	24.38	23.32	24.85	19.65	24.60	25.33
	IEN+FC $_{\tilde{w}}$	23.62	23.03	24.44	19.11	25.44	27.44
	Maxout	26.34	24.12	24.72	24.65	26.89	29.29
	Base	24.13	22.26	24.85	19.86	24.76	26.02

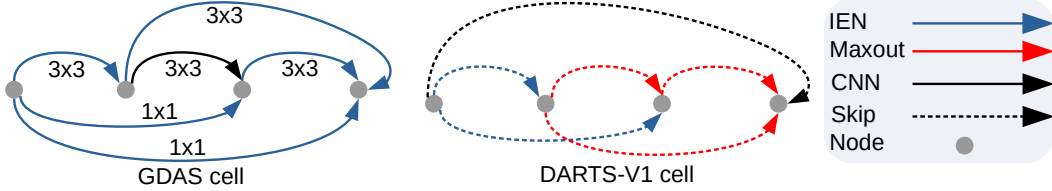


Figure 4: Cells discovered using NAS DARTS-V1 and GDAS search algorithms. 1×1 and 3×3 are the CNN kernel sizes. IEN and maxout have $m = 4$. The cells are acyclic graphs. The dashed arrows are skip connections. Colors represent different approaches.

4.3 Extension to NAS

In order to make sure that IENs are preferred in deep model design, we tested it using NAS search algorithms. We introduced IEN and maxout as operators in both DARTS-V1 [24] and GDAS [8] search algorithms besides regular CNNs. We ran NAS-Bench-201 [9] on both DARTS and GDAS to discover the cells. We report results on CIFAR-10, CIFAR-100 and ImageNet-16-120. ImageNet-16-120 is a subset of ImageNet [34] that is used by NAS-Bench-201. From Figure 4 GDAS shows a high preference towards using IENs. DARTS-V1 resulted in a mix between maxout and IEN and used IEN in the early stage of the cell. This early usage of IEN might correlate with the importance of reducing the variance in the early layers of a deep model. Also, DARTS resulted in a skip-connect cell which is a known bias in DARTS algorithm [9]. Table 3 shows that our discovered cell which depends on IEN resulted in a better performance than cells that use regular CNNs.

Table 3: Accuracies of four cell models based on NAS-Bench-201 configuration, using our discovered cells and the original NAS-Bench-201 cells.

NAS Search Method	Cell	CIFAR-10	CIFAR-100	ImageNet-16-120
DARTS-V1	Ours	41.28	17.36	16.77
	NAS-Bench-201 Cell	39.77	15.03	16.43
GDAS	Ours	91.26$\pm$0.11	72.68$\pm$0.12	47.6$\pm$0.33
	NAS-Bench-201 Cell	89.89 $\pm$ 0.08	71.34 $\pm$ 0.04	41.59 $\pm$ 1.33

5 Conclusion

We presented IENs which enhance the performance of a deep model by utilizing the concept of ensembles while keeping the model parameters free from the excess ensemble weights. We theoretically analyzed the variance reduction gain of IEN and other approaches with respect to ordinary deep model variance. We empirically showed that IENs outperformed other methods when applied on known deep models architectures. We analyzed IEN behavior versus that of outer ensembles and extended it to NAS with the discovery of new cells. In the future, we want to enhance IEN training mechanism to only use the same number of parameters as the base model in the training phase.

References

- [1] Kaiming he initialization. https://pytorch.org/docs/stable/nn.init.html#torch.nn.init.calculate_gain. Accessed: 2020-06-10.
- [2] Kaiming he initialization. <https://medium.com/@shoray.goel/kaiming-he-initialization-a8d9ed0b5899>. Accessed: 2019-07-14.
- [3] H. Abbasiyan, C. Drummond, N. Japkowicz, and S. Matwin. Inner ensembles: Using ensemble methods inside the learning algorithm. In *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, pages 33–48. Springer, 2013.
- [4] J. Ba and R. Caruana. Do deep nets really need to be deep? In *Advances in neural information processing systems*, pages 2654–2662, 2014.
- [5] P. Baldi and P. J. Sadowski. Understanding dropout. In C. J. C. Burges, L. Bottou, M. Welling, Z. Ghahramani, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 26*, pages 2814–2822. Curran Associates, Inc., 2013.
- [6] T. G. Dietterich. Ensemble methods in machine learning. In *International workshop on multiple classifier systems*, pages 1–15. Springer, 2000.
- [7] J. Ding, R. Eldan, A. Zhai, et al. On multiple peaks and moderate deviations for the supremum of a gaussian field. *The Annals of Probability*, 43(6):3468–3493, 2015.
- [8] X. Dong and Y. Yang. Searching for a robust neural architecture in four gpu hours. In *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 1761–1770, 2019.
- [9] X. Dong and Y. Yang. Nas-bench-201: Extending the scope of reproducible neural architecture search. In *International Conference on Learning Representations (ICLR)*, 2020.
- [10] H. Drucker, C. Cortes, L. D. Jackel, Y. LeCun, and V. Vapnik. Boosting and other ensemble methods. *Neural Computation*, 6(6):1289–1301, 1994.
- [11] emcor (<https://stats.stackexchange.com/users/48639/emcor>). Extreme value theory - show: Normal to gumbel. Cross Validated. URL:<https://stats.stackexchange.com/q/105745> (version: 2015-05-17).
- [12] C.-Y. Fu. pytorch-vgg-cifar10. <https://github.com/chengyangfu/pytorch-vgg-cifar10>. Accessed: 2020-05-26.
- [13] S. Geman, E. Bienenstock, and R. Doursat. Neural networks and the bias/variance dilemma. *Neural computation*, 4(1):1–58, 1992.
- [14] X. Glorot and Y. Bengio. Understanding the difficulty of training deep feedforward neural networks. In Y. W. Teh and M. Titterton, editors, *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics*, volume 9 of *Proceedings of Machine Learning Research*, pages 249–256, Chia Laguna Resort, Sardinia, Italy, 13–15 May 2010. PMLR.
- [15] I. J. Goodfellow, D. Warde-Farley, M. Mirza, A. Courville, and Y. Bengio. Maxout networks. *arXiv preprint arXiv:1302.4389*, 2013.
- [16] J. Hartung, G. Knapp, and B. K. Sinha. *Statistical meta-analysis with applications*, volume 738. John Wiley & Sons, 2011.
- [17] K. He, X. Zhang, S. Ren, and J. Sun. Deep residual learning for image recognition. *CoRR*, abs/1512.03385, 2015.
- [18] K. He, X. Zhang, S. Ren, and J. Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the 2015 IEEE International Conference on Computer Vision (ICCV)*, ICCV ’15, page 1026–1034, USA, 2015. IEEE Computer Society.
- [19] G. Huang, Z. Liu, and K. Q. Weinberger. Densely connected convolutional networks. *CoRR*, abs/1608.06993, 2016.
- [20] Y. Idelbayev. Proper ResNet implementation for CIFAR10/CIFAR100 in PyTorch. https://github.com/akamaster/pytorch_resnet_cifar10. Accessed: 20xx-xx-xx.
- [21] A. Krizhevsky and G. Hinton. Learning multiple layers of features from tiny images. 2009.

- [22] A. Krogh and J. Vedelsby. Neural network ensembles, cross validation, and active learning. In *Advances in neural information processing systems*, pages 231–238, 1995.
- [23] M. Leadbetter and H. Rootzen. Extremal theory for stochastic processes. *The Annals of Probability*, pages 431–478, 1988.
- [24] H. Liu, K. Simonyan, and Y. Yang. DARTS: differentiable architecture search. *CoRR*, abs/1806.09055, 2018.
- [25] D. Opitz and R. Maclin. Popular ensemble methods: An empirical study. *Journal of artificial intelligence research*, 11:169–198, 1999.
- [26] M. Opitz, H. Possegger, and H. Bischof. Efficient model averaging for deep neural networks. In S.-H. Lai, V. Lepetit, K. Nishino, and Y. Sato, editors, *Computer Vision – ACCV 2016*, pages 205–220, Cham, 2017. Springer International Publishing.
- [27] S. Park and N. Kwak. Analysis on the dropout effect in convolutional neural networks. In *Asian conference on computer vision*, pages 189–204. Springer, 2016.
- [28] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, et al. Pytorch: An imperative style, high-performance deep learning library. In *Advances in Neural Information Processing Systems*, pages 8024–8035, 2019.
- [29] G. Pleiss, D. Chen, G. Huang, T. Li, L. van der Maaten, and K. Q. Weinberger. Memory-efficient implementation of densenets. *arXiv preprint arXiv:1707.06990*, 2017.
- [30] X. Qiu, L. Zhang, Y. Ren, P. N. Suganthan, and G. Amaratunga. Ensemble deep learning for regression and time series forecasting. In *Computational Intelligence in Ensemble Learning (CIEL), 2014 IEEE Symposium on*, pages 1–6. IEEE, 2014.
- [31] P. Rajpurkar, J. Zhang, K. Lopyrev, and P. Liang. Squad: 100,000+ questions for machine comprehension of text. In *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, 2016.
- [32] L. Rokach. Ensemble-based classifiers. *Artificial Intelligence Review*, 33(1-2):1–39, 2010.
- [33] B. E. Rosen. Ensemble learning using decorrelated neural networks. *Connection science*, 8(3-4):373–384, 1996.
- [34] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei. ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision (IJCV)*, 115(3):211–252, 2015.
- [35] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, et al. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3):211–252, 2015.
- [36] K. Simonyan and A. Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.
- [37] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov. Dropout: A simple way to prevent neural networks from overfitting. *Journal of Machine Learning Research*, 15(56):1929–1958, 2014.
- [38] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich. Going deeper with convolutions. In *Computer Vision and Pattern Recognition (CVPR)*, 2015.
- [39] user_lambda (https://math.stackexchange.com/users/338905/user_lambda). Distribution of the maximum of a large number of normally distributed random variables. Mathematics Stack Exchange. URL:<https://math.stackexchange.com/q/2035079> (version: 2016-11-29).
- [40] A. Veit, M. J. Wilber, and S. Belongie. Residual networks behave like ensembles of relatively shallow networks. In *Advances in neural information processing systems*, pages 550–558, 2016.
- [41] A. Wasay, B. Hentschel, Y. Liao, S. Chen, and S. Idreos. Mothernets: Rapid deep ensemble learning. In *Proceedings of the 3rd MLSys Conference (MLSys)*, 2020.
- [42] S. Xie, R. Girshick, P. Dollár, Z. Tu, and K. He. Aggregated residual transformations for deep neural networks. In *Computer Vision and Pattern Recognition (CVPR), 2017 IEEE Conference on*, pages 5987–5995. IEEE, 2017.
- [43] Z.-H. Zhou. *Ensemble methods: foundations and algorithms*. Chapman and Hall/CRC, 2012.

- [44] Z.-H. Zhou, J. Wu, and W. Tang. Ensembling neural networks: many could be better than all. *Artificial intelligence*, 137(1-2):239–263, 2002.
- [45] B. Zoph and Q. V. Le. Neural architecture search with reinforcement learning. *CoRR*, abs/1611.01578, 2016.

A Lemmas

A.1 Variance of the product of independent random variables

if $y_1, y_2, \dots, y_n$ are independent random variables, then:

$$\text{Var}[y_1 * y_2 \dots y_n] = \prod_{i=1}^n (\text{Var}[y_i] + \mathbb{E}[y_i]^2) - \prod_{i=1}^n \mathbb{E}[y_i]^2 \quad (22)$$

A.2 Variance gain of a non-linear activation function

Let y be a random variable, $x = f(y)$, where f is a nonlinear activation function, one can state that:

$$\text{Var}[x] = \int_{-\infty}^{\infty} f(y)^2 \mathbb{P}(y) dy \quad (23)$$

By using first order Taylor series expansion:

$$\text{Var}[x] \approx \int_{-\infty}^{\infty} (f(y_0) + f'(y_0)(y - y_0))^2 \mathbb{P}(y) dy \quad (24)$$

Where $y_0 = 0$, then:

$$\text{Var}[x] \approx \int_{-\infty}^{\infty} f(0)^2 \mathbb{P}(y) dy + \int_{-\infty}^{\infty} f'(0)^2 y^2 \mathbb{P}(y) dy + 2 \int_{-\infty}^{\infty} f(0) f'(0) y \mathbb{P}(y) dy \quad (25)$$

While Equation 25 is valid only for smoothness C1 functions it also works reasonably on non C1 functions like ReLU = $\max(0, y)$. If we choose $x = \max(0, y)$ aka a ReLU function, we have:

$$x'(y) = \begin{cases} 1 & \text{if } y \geq 0 \\ 0 & \text{if } y < 0 \end{cases} \quad (26)$$

By using Equation 25 and by assuming x has zero mean with symmetric distribution:

$$\text{Var}[x] \approx \int_{-\infty}^{\infty} f'(0)^2 y^2 \mathbb{P}(y) dy = \int_0^{\infty} y^2 \mathbb{P}(y) dy \quad (27)$$

Thus, we can state:

$$\text{Var}[x] \approx \frac{1}{2} \int_{-\infty}^{\infty} y^2 \mathbb{P}(y) dy = \beta^2 \text{Var}[y] \quad (28)$$

We denote the gain $\frac{1}{2}$ as β^2 . Same procedure can be applied to different activation functions. A full table of different gains for commonly used deep model activation function can be found at [1].

Another approach for ReLU gain is [2]:

$$\begin{aligned} \text{Var}[x] &= \int_{-\infty}^{\infty} x^2 \mathbb{P}(x) dx \\ \text{Var}[x] &= \int_{-\infty}^{\infty} \max(0, y)^2 \mathbb{P}(y) dy \\ \text{Var}[x] &= \int_0^{\infty} y^2 \mathbb{P}(y) dy \\ \text{Var}[x] &= \frac{1}{2} * \int_{-\infty}^{\infty} y^2 \mathbb{P}(y) dy \\ \text{Var}[x] &= \frac{1}{2} * \text{Var}[y] \\ \text{Var}[x] &= \beta^2 * \text{Var}[y] \end{aligned}$$

B Dropout variance response

$$\mathbf{y}_l^{\text{dropout}} = \Delta_l \circ \mathbf{w}_l \mathbf{x}_l$$

Assumptions:

- $\mathbf{w}_l = \{w_{ji} : j \in n, i \in d\} \in \mathbb{R}^{n \times d}$ elements are initialized to be mutually independent with zero mean and a symmetric distribution
- $\mathbf{x}_l = \{x_i : i \in d\} \in \mathbb{R}^{d \times 1}$ elements are initialized to be mutually independent sharing the same distribution with zero mean
- $\Delta_l = \{\delta_i : i \in n\} \in \mathbb{R}^{n \times 1}$ elements are Bernoulli δ_l mutually independent random variables which are multiples element wise by $\mathbf{w}_l$
- $\mathbf{w}_l$, Δ_l and $\mathbf{x}_l$ are independent from each other
- $\mathbf{y}_l^{\text{dropout}} = \{y_i : i \in n\} \in \mathbb{R}^{n \times 1}$

The variance of each $y_l^{\text{dropout}} \in \mathbf{y}_l^{\text{dropout}}$, where l is the layer number, is:

$$\text{Var}[y_l^{\text{dropout}}] = n_l \text{Var}[\delta_l w_l x_l] \quad (29)$$

By using the lemma in A.1:

$$\text{Var}[y_l^{\text{dropout}}] = n_l [(\text{Var}[\delta_l] + \mathbb{E}[\delta_l]^2)(\text{Var}[w_l] + \mathbb{E}[w_l]^2)(\text{Var}[x_l] + \mathbb{E}[x_l]^2) - (\mathbb{E}[\delta_l]^2)(\mathbb{E}[w_l]^2)(\mathbb{E}[x_l]^2)] \quad (30)$$

$$= n_l [(\text{Var}[\delta_l] + \mathbb{E}[\delta_l]^2)(\text{Var}[w_l] + \mathbb{E}[w_l]^2)(\text{Var}[x_l] + \mathbb{E}[x_l]^2) - (\mathbb{E}[\delta_l]^2)(\mathbb{E}[w_l]^2)(\mathbb{E}[x_l]^2)] \quad (31)$$

Thus,

$$\text{Var}[y_l^{\text{dropout}}] = n_l (\text{Var}[\delta_l] + \mathbb{E}[\delta_l]^2) \text{Var}[w_l] \text{Var}[x_l] \quad (32)$$

By using the the results and assumption made at A.2 given that $x_l = f(y_{l-1})$, we have:

$$\text{Var}[y_l^{\text{dropout}}] = n_l \beta^2 (\text{Var}[\delta_l] + \mathbb{E}[\delta_l]^2) \text{Var}[w_l] \text{Var}[y_{l-1}] \quad (33)$$

For cascaded l layers and with the fact that $\text{Var}[y_1]$ represents the variance of the input layer, we arrive at:

$$\text{Var}[y_L^{\text{dropout}}] = \text{Var}[y_1] \left(\prod_{l=2}^L \beta^2 n_l (\text{Var}[\delta_l] + \mathbb{E}[\delta_l]^2) \text{Var}[w_l] \right) \quad (34)$$

For a Bernoulli random variable the maximum $\mathbb{E}[\delta_l] = \frac{1}{2}$ with $\text{Var}[\delta_l] = \frac{1}{4}$, then we can upper bound the $\text{Var}[y_L^{\text{dropout}}]$ to be:

$$\text{Var}[y_L^{\text{dropout}}] \leq \text{Var}[y_1] \left(\prod_{l=2}^L \beta^2 n_l \left(\frac{1}{4} + \left(\frac{1}{2} \right)^2 \right) \text{Var}[w_l] \right) \quad (35)$$

$$\text{Var}[y_L^{\text{dropout}}] \leq \text{Var}[y_1] \left(\prod_{l=2}^L \beta^2 n_l \frac{1}{2} \text{Var}[w_l] \right) \quad (36)$$

C Maxout variance upper bound

$$\mathbf{y}_l^{\text{maxout}} = \max\{\mathbf{w}_l^i \mathbf{x}_l\}_i^m$$

Assumptions:

- $\mathbf{w}_l^i = \{w_{kj}^i : k \in n, j \in d\} \in \mathbb{R}^{n \times d}$ elements are initialized to be mutually independent with zero mean and a symmetric distribution
- $\mathbf{x}_l = \{x_i : i \in d\} \in \mathbb{R}^{d \times 1}$ elements are initialized to be mutually independent sharing the same distribution with zero mean
- $\mathbf{w}_l$ and $\mathbf{x}_l$ are independent from each other
- $\mathbf{y}_l^{\text{maxout}} = \{y_i : i \in n\} \in \mathbb{R}^{n \times 1}$

The variance of each $y_l^{\text{maxout}} \in \mathbf{y}_l^{\text{maxout}}$, where l is the layer number, is:

$$\text{Var}[y_l^{\text{maxout}}] = n_l \text{Var}[\max\{w_l^i x_l\}_i^m] \quad (37)$$

This can be upper bounded using the fact the $\text{Var}[\max(z_i)_i^m] \leq \text{Var}[\sum_{i \in m} z_i]$, applying this bound:

$$\text{Var}[y_l^{\text{maxout}}] \leq n_l \text{Var}[\sum_{m \in i} \{w_l^i x_l\}] = n_l \sum_{m \in i} \text{Var}[w_l^i x_l] \quad (38)$$

Because w_l^i and x_l are both independent and by using lemma A.1:

$$\text{Var}[y_l^{\text{maxout}}] \leq n_l \sum_{m \in i} [(\text{Var}[w_l^i] + \mathbb{E}[w_l^i]^2)(\text{Var}[x_l] + \mathbb{E}[x_l]^2) - (\mathbb{E}[w_l^i]^2)(\mathbb{E}[x_l]^2)] \quad (39)$$

$$= n_l \sum_{m \in i} [(\text{Var}[w_l^i] + \cancel{\mathbb{E}[w_l^i]^2}^0)(\text{Var}[x_l] + \cancel{\mathbb{E}[x_l]^2}^0) - (\cancel{\mathbb{E}[w_l^i]^2}^0)(\cancel{\mathbb{E}[x_l]^2}^0)] \quad (40)$$

$$= n_l \sum_{m \in i} \text{Var}[w_l^i] \text{Var}[x_l] \quad (41)$$

By using the the results and assumption made at A.2 given that $x_l = f(y_{l-1})$, we have:

$$\text{Var}[y_l^{\text{maxout}}] \leq n_l \beta^2 \text{Var}[y_{l-1}] \sum_{m \in i} \text{Var}[w_l^i] \quad (42)$$

Because all elements of w_l^i share the same distribution, their variance is equal, we can donate $\text{Var}[w_l^i] = \text{Var}[w_l]$ and by their independence:

$$\text{Var}[y_l^{\text{maxout}}] \leq n_l m \beta^2 \text{Var}[y_{l-1}] \text{Var}[w_l] \quad (43)$$

For cascaded l layers and with the fact that $\text{Var}[y_1]$ represents the variance of the input layer, we arrive at:

$$\text{Var}[y_L^{\text{maxout}}] \leq \text{Var}[y_1] \left(\prod_{l=2}^L n_l m \beta^2 \text{Var}[w_l] \right) \quad (44)$$

D Maxout variance lower bound

$$\mathbf{y}_l^{\text{maxout}} = \max\{\mathbf{w}_l^i \mathbf{x}_l\}_i^m$$

Assumptions:

- $\mathbf{w}_l^i = \{w_{kj}^i : k \in n, j \in d\} \in \mathbb{R}^{n \times d}$ elements are initialized to be mutually independent with zero mean and a symmetric distribution
- $\mathbf{x}_l = \{x_i : i \in d\} \in \mathbb{R}^{d \times 1}$ elements are initialized to be mutually independent sharing the same distribution with zero mean
- $\mathbf{w}_l$ and $\mathbf{x}_l$ are independent from each other
- $(w_l^i x_l) \sim \text{iid } \mathcal{N}(0, 1)$
- $\mathbf{y}_l^{\text{maxout}} = \{y_i : i \in n\} \in \mathbb{R}^{n \times 1}$

From [7] for $i \in m$ the $\text{Var}[\max_i z_i] \geq \frac{c}{\log m}$, where $z_i \sim \mathcal{N}(0, 1)$ and $c > 0$. Then:

$$\text{Var}[y_l^{\text{maxout}}] \geq n_l \frac{c}{\log m} \quad (45)$$

E Maxout variance asymptotic bound

$$\mathbf{y}_l^{\text{maxout}} = \max\{\mathbf{w}_l^i \mathbf{x}_l\}_i^m$$

Assumptions:

- $\mathbf{w}_l^i = \{w_{kj}^i : k \in n, j \in d\} \in \mathbb{R}^{n \times d}$ elements are initialized to be mutually independent with zero mean and a symmetric distribution
- $\mathbf{x}_l = \{x_i : i \in d\} \in \mathbb{R}^{d \times 1}$ elements are initialized to be mutually independent sharing the same distribution with zero mean
- $\mathbf{w}_l$ and $\mathbf{x}_l$ are independent from each other
- $(w_l^i x_l) \sim \text{iid } \mathcal{N}(0, \sigma^2)$
- $\mathbf{y}_l^{\text{maxout}} = \{y_i : i \in n\} \in \mathbb{R}^{n \times 1}$

The variance of each $y_l^{\text{maxout}} \in \mathbf{y}_l^{\text{maxout}}$, where l is the layer number, is:

$$\text{Var}[y_l^{\text{maxout}}] = n_l \text{Var}[\max\{w_l^i x_l\}_i^m] \quad (46)$$

For more details about this proof please check [39, 11, 23].

Recall that the standard normal CDF Φ is such that:

$$1 - \Phi(y) \sim \frac{e^{-y^2/2}}{y\sqrt{2\pi}} \quad (47)$$

when $y \rightarrow \infty$ and that, for every y :

$$P(Y_n \leq \mu + y\sigma) = \Phi(y)^m \quad (48)$$

Hence, if y_m is chosen such that:

$$m \frac{e^{-y_n^2/2}}{y_n\sqrt{2\pi}} = t \quad (49)$$

for some given t , then:

$$P(Y_m \leq \mu + y_m\sigma) \rightarrow e^{-t} \quad (50)$$

Solving this for y_m yields:

$$y_m = \sqrt{2 \ln m} - \frac{\ln \ln m + 2 \ln t + \ln(4\pi)}{2\sqrt{2 \ln m}} \quad (51)$$

Then, we can define:

$$Z_m = \sqrt{2 \ln m} \frac{Y_m - \mu}{\sigma} - 2 \ln m + \frac{1}{2} \ln \ln m + \frac{1}{2} \ln(4\pi) \quad (52)$$

where, for every real z :

$$P(Z \leq z) = \exp(-e^{-z}) = G(\mu = 0, \beta = 1) \text{ a standard Gumbel distribution.} \quad (53)$$

Which aligns with the convergence from extreme value theorem. From Equation 52, for $\mu = 0$, we have and for sufficiently large m :

$$Y_m \approx \frac{\sigma}{\sqrt{2 \ln m}} [Z_m + 2 \ln m - \frac{1}{2} \ln \ln m - \frac{1}{2} \ln(4\pi)] \quad (54)$$

Hence:

$$\text{Var}[Y_m] \approx \text{Var}\left[\frac{\sigma}{\sqrt{2 \ln m}} [Z_m + 2 \ln m - \frac{1}{2} \ln \ln m - \frac{1}{2} \ln(4\pi)]\right] \quad (55)$$

$$\text{Var}[Y_m] \approx \frac{\sigma^2}{2 \ln m} \text{Var}[Z_m] \quad (56)$$

A standard Gumbel distribution have a variance of $\frac{\pi^2}{6}$, then:

$$\text{Var}[Y_m] \approx \frac{\pi^2}{6} \frac{\sigma^2}{2 \ln m} \quad (57)$$

In our case $Y_m = y_l^{\text{maxout}}$ and $\text{Var}[w_l x_l] = \sigma^2$, yielding:

$$\text{Var}[y_l^{\text{maxout}}] \approx n_l \frac{\pi^2}{6} \frac{\text{Var}[w_l x_l]}{2 \ln m} = n_l \frac{\pi^2}{6} \frac{\sigma^2}{2 \ln m} \quad (58)$$

F Models training time and parameters count

Table 4: Model parameters count / Average training time per epoch in seconds. M stands for Million, K stands for Thousand. All models were trained on Nvidia V100 GPU. Model in blue have the same size at the testing time. +FC stands for IEN applied on both FC and CNN layers. Maxout results are only on CNN layers. All models have $m = 4$

Dataset		ResNet56	ResNet110	DenseNet40-12	DenseNet100-12	VGG16	VGG19
CIFAR-10	IEN	4.25M / 55.16	8.60M / 97.17	854.94K / 102.79	3.74M / 239.49	74.11M / 26.62	100.66M / 32.34
	IEN+FC	4.25M / 55.9	8.61M / 105.38	860.26K / 101.01	3.75M / 244.74	76.23M / 27.46	102.79M / 32.93
	Maxout	4.25M / 57.45	8.60M / 110.53	854.94K / 106.33	3.74M / 244.74	74.11M / 28.20	100.66M / 33.98
	Base	853.02K / 17.16	1.73M / 42.01	176.12 / 51.96	769.16k / 115.16	15.25M / 7.61	20.57M / 9.21
CIFAR-100	IEN	4.25M / 58.56	8.61M / 94.75	866.91K / 96.13	3.77M / 239.63	74.16M / 26.68	100.71M / 32.29
	IEN+FC	4.28M / 58.30	8.64M / 96.38	920.11K / 98.71	3.75M / 244.74	76.46M / 27.32	103.02M / 33.10
	Maxout	4.25M / 62.64	8.61M / 111.1	866.91K / 104.53	3.77M / 253.85	74.16M / 28.18	100.71M / 34.02
	Base	858.87k / 20.01	1.73M / 36.28	188.09K / 47.73	800.3K / 114.13	15.30M / 7.62	20.61M / 9.20

From Table 4 we notice that Maxout training time is always more than IEN. There is also an increase in both training time and parameters size of IEN+FC against IEN only. Training time of IEN with $m = 4$ is much cheaper than training 4 ordinary models, but it comes on the cost of the memory only during the training time.

G Empirical analysis of dropout

Table 5: Results of applying dropout everywhere except the input and output layer. Mean error rate and std for multiple runs are reported. The lower the better. IEN uses $m = 4$.

Dataset		IEN	Base	Dropout	Dropout+IEN
CIFAR-10	ResNet56	6.93±0.26	8.38±1.2	90.48±0.04	90.12±0.01
	ResNet110	6.27±0.32	6.38±0.48	91.29±0.07	90.13±0.01
CIFAR-100	ResNet56	29.34±0.49	29.97±0.71	99.09±0.015	98.98±0.01
	ResNet110	28.17±0.11	27.83±0.64	98.91±0.01	98.98±0.01

We wanted to analysis the effect of using dropout everywhere except the input and output layers. Dropout performance is the best when it is probability set to 0.5. We tested two combinations, dropout applied as before and IEN followed by dropout. Table 5 summarizes these experiments. We notice that dropout only leads to very bad performance in comparison with IEN and base models. This states that even theoretically dropout can lower the variance, yet it is not applicable everywhere unlike IEN. Also, from the same Table 5 we notice that when IEN is combined with dropout the results get slightly better. This suggests the power of IEN in reducing the variance even when the model is being driven by another component.

H Guide to the code

Most of the models needed 1x Nvidia V100 with 32 GB of GPU memory for training. Some DenseNet models needed 2x Nvidia V100. Our code is attached for results reproduction with a quick README.MD.

H.1 ResNet Details

We used an open source repository [20] that produces the same results like ResNet original paper using Pytorch [28]. The models were trained for 200 epochs using a learning rate of 0.1 and batch size of 128.

H.2 DenseNet Details

We used an open source repository [29] that produces the same results like original DenseNet paper but in more memory efficient way using Pytorch. The models were trained for 300 epochs using a learning rate of 0.1 and batch size of 64. The is code located at:

H.3 VGG Details

We used an open source repository [12]. The models were trained for 300 epochs using a learning rate of 0.9 and batch size of 128.

H.4 NAS Details

We used the default settings in NAS-Bench-201 [9]. We only added our IEN and Maxout operators and searched, trained the models.