Error Resilience Analysis for Systematically Employing Approximate Computing in Convolutional Neural Networks

Muhammad Abdullah Hanif* , Rehan Hafiz† and Muhammad Shafique*
* Institute of Computer Engineering, Vienna University of Technology, Vienna, Austria
† Department of Electrical Engineering, Information Technology University, Lahore, Pakistan
Emails: {muhammad.hanif, muhammad.shafique}@tuwien.ac.at, rehan.hafiz@itu.edu.pk

***Abstract*—Approximate computing is an emerging paradigm for error resilient applications as it leverages accuracy loss for improving power, energy, area, and/or performance of an application. The spectrum of error resilient applications includes the domains of Image and video processing, Artificial Intelligence (AI) and Machine Learning (ML), data analytics, and other Recognition, Mining, and Synthesis (RMS) applications. In this work, we address one of the most challenging question, i.e., how to systematically employ approximate computing in Convolution Neural Networks (CNNs), which are one of the most compute-intensive and the pivotal part of AI. Towards this, we propose a methodology to systematically analyze error resilience of deep CNNs and identify parameters that can be exploited for improving performance/efficiency of these networks for inference purposes. We also present a case study for significance-driven classification of filters for different convolutional layers, and propose to prune those having the least significance, and thereby enabling accuracy vs. efficiency tradeoffs by exploiting their resilience characteristics in a systematic way.**

I. Introduction

Convolutional Neural Networks have become the new normal in Machine Learning (ML) and Artificial Intelligence (AI) because of their high achievable accuracy in solving non-trivial classification [1], recognition [2], and/or cognition problems. This enhanced performance generally comes at the cost of increased computational and memory requirements resulting from increased number of convolutional/fully-connected layers [3] [4] and increased number of filters/neurons per layer [5]. While this increased computational complexity has provided a significant improvement in performance, the increased hardware requirements for training and inference (testing) is drifting these networks away from being deployed in real world applications involving resource constraint devices, especially portable multimedia devices. There are two main approaches that can be employed for alleviating this problem: 1) Increase computational power of the processing devices, which in case of resource constraint portable devices is not a feasible solution because of the battery constraints. 2) Reduce the computational complexity of such networks at the cost of insignificant accuracy loss by employing optimization strategies so that the state-of-the-art high accuracy networks can easily be ported to resource constraint devices.

A huge body of work is available on optimizing over-parameterized neural networks mainly using pruning [6], weight sharing [7], and quantization [8] to achieve better computational/storage efficiency. Combination of these techniques have also been explored in [9] to significantly compress the size of NNs. Apart, from the aforementioned techniques approximate components like Approximate Adders [10], Multipliers [11], and Multiply and Accumulate operators (MACs) [12] have also been employed for boosting the efficiency of underlying hardware implementing the neural networks.

As mentioned above, a number of optimization/approximation techniques are available that can be employed for alleviating the computational and memory requirements of compute and memory intensive neural networks. However, in almost all the works, these techniques have been studied individually, i.e., independent of other optimizations. The complete set of approximation and optimization knobs that can be exploited collectively for achieving optimal/near-optimal gains in energy, power, and/or execution time are: 1) Weights/Filter pruning [13], 2) Weight sharing [7], 3) Fixed point conversion and quantization [8], and 4) Approximate components for realizing underlying hardware [14].

To employ such techniques, there is a dire need for error resilience analysis that can estimate the possible amount of approximations that can be employed and/or can identify the possible candidates that can be removed/optimized without having a significant impact on the overall accuracy of the system.

A. *Novel Contribution*

In this work we propose:

- A systematic methodology for employing approximations/optimizations both at software (for pruning ineffectual/redundant parameters) as well as at hardware levels (for further alleviating the computational and memory requirements by employing quantization).
- A methodology to systematically analyze error-resilience of a Deep-CNN, individually at software as well as hardware level.
- We also present a case study for significance-driven classification of filters for different convolutional layers, and propose to prune those having the least significance, and thereby enabling accuracy vs. efficiency tradeoffs by exploiting their resilience characteristics in a systematic way.

Rest of the paper is organized as follows: section II introduces the terminologies and a few basics related to CNNs. Section III presents the methodology for employing approximations in convolutional neural networks. Section IV and V discusses the error analysis while section VI concludes the paper.

II. Preliminaries

This section briefly introduces the terminologies and basics of convolutional neural networks that will be used in the forthcoming sections.

Fig. 1 illustrates the VGG-f architecture [15] that (without any loss of generality) we used for simulations and analysis in the following sections. The architecture is primarily composed of 8 computational layers (5 convolutional and 3 fully-connected). However, In convolutional neural networks the Convolutional (CONV) layers are the most compute intensive layers and therefore for the current work we focused our attention on pruning and approximating only the convolutional layers in a CNN.

Fig. 2 provides a detailed view of an i^{th} convolutional layer where input feature maps (denoted by fm_i) are convolved with f_i filters to produce the output feature maps (fm_{i+1}). w_i and h_i denotes the width and height of the input feature maps, respectively, while x_i denotes the number of channels. The filters in a convolutional layer usually have same width and height (illustrated by k_i in fig.

2) while the channels are equivalent to the number of channels of the input feature maps, i.e., x_i. However, the number of channels in the output feature maps are equivalent to the number of filters N_i in the i^{th} convolutional layers. Using the parameters defined above, we can compute the total number of Multiply and Accumulate (MAC) operations (from henceforth refereed to as computations) required for computing a single channel of output feature map fm_{i+1}, which is given as $h_{i+1} \times w_{i+1} \times k^2 \times x_i$. Hence, the total number of computations required for computing the complete output feature maps is x_{i+1} times the number of computations required for computing a single channel.

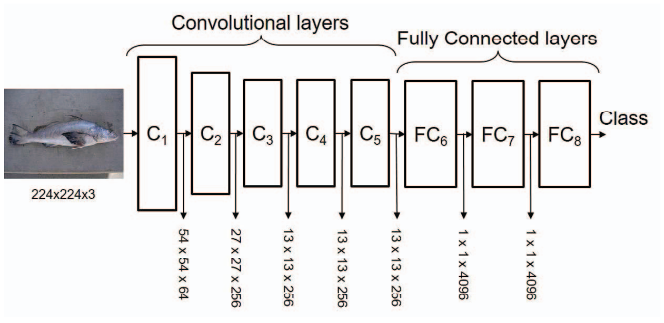

Fig. 1: VGG-f architecture. For illustrative purposes the activation, normalization, and pooling layers are not shown.

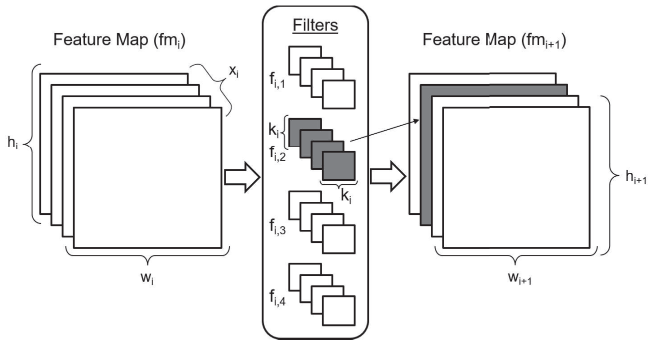

Fig. 2: Illustrative view of a convolutional layer

III. Methodology

In order to employ approximations in a CNN we propose to use the following steps (also illustrated in fig. 3):

1) Analyze the error resilience of the CNN to identify the potential for a specific approximation.
2) Apply resilience aware approximations by either pruning the least significant parameters or by utilizing approximate hardware for improving the performance/energy efficiency.
3) Re-train the network to re-gain a fraction of the accuracy lost because of approximations (Optional).
4) Re-evaluate the accuracy of the approximated network to decide whether to re-iterate the process or not.

However, for this paper we limit the scope of our work to error resilience evaluation of CNNs.

IV. Error Resilience Analysis of CNNs

Different approximations introduce different types of errors and in order to analyze the resilience of a network against all types of errors we divide the error resilience analysis step into two major classes: 1)

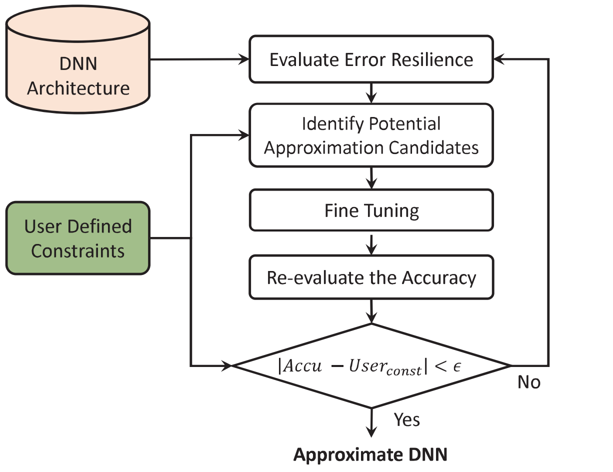
DNN Architecture
User Defined Constraints
Evaluate Error Resilience
Identify Potential Approximation Candidates
Fine Tuning
Re-evaluate the Accuracy
$|Accu - User_{const}| < \epsilon$
No
Yes
Approximate DNN

Fig. 3: A methodology for employing approximations in convolutional neural networks

hardware level 2) software level error resilience analysis. However, in this section, we present only the hardware level error resilience analysis and the software level error resilience analysis based upon filter pruning is presented as a case study in section V.

A. Hardware level Error Resilience analysis methodology

There are two possible hardware approximation knobs that can be exploited for improving the efficiency of a Deep-CNN 1) Quantization: where the floating point operations are transformed to fixed point and, to improve the overall memory and energy efficiency even further, the word sizes of the activations and weights are reduced. 2) Along side quantization, approximate hardware components, e.g., approximate adders, multipliers, and memory units, can also be used to reduce the overall energy/latency of a hardware. In both the aforementioned cases, errors are introduced at multiple locations in a network. Therefore, by assuming the error sources to be independent and identically distributed (i.i.d.), we can simulate the error resilience of a network by introducing Random Gaussian or White Gaussian Noise (RGN or WGN) at particular locations in a network. This is because of the fact that errors from multiple sources when added together generates a Gaussian distribution [16]. Therefore, to mimic this behavior, we employed WGN, shown in fig. 4.

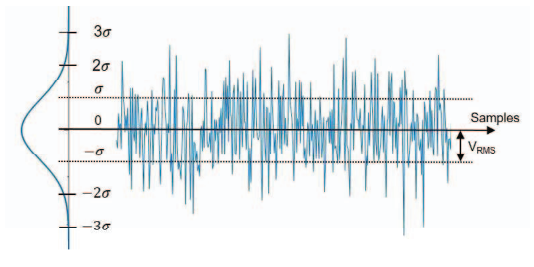

Fig. 4: White Gaussian Noise (WGN)

Fig. 5 illustrates the effects of introducing WGN individually at the output of convolutional layers on the output accuracy of the network. This is done in order to mimic the behavior of using approximate computing inside the respective convolutional layers. For this analysis, we used VGG-f architecture for classifying 5000 images (randomly selected from the ImageNet validation set [17]). The accuracy reported in the results is Top-5 accuracy, which means that the network is given credit for every image for which the correct class appeared in Top-5 recognized classes. Fig. 5 also illustrates that a network have a different level of error tolerance for the same error in different convolutional layers. For example, when WGN of intensity 30dBW was introduced at the output of CONV-layer 2 it reduced the accuracy of the network to less than 10%, however, when the same intensity of noise was introduced at the output of CONV-layer 4 the network maintained its accuracy around 70%. This varying error tolerance of the computational layers, for the same error magnitudes, can be exploited for further improving the performance/energy efficiency of a network.

We also analyzed the effect of biased noise, i.e., noise with non-zero mean, on the classification accuracy of the network and observed that error distributions having zero-mean produces better results, as illustrated in fig. 6. For this experiment, we used a WGN of intensity 20dBW + various bias values.

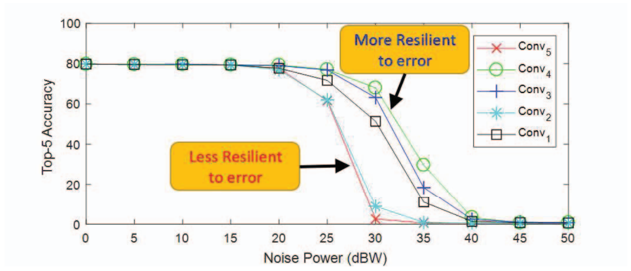

Fig. 5: Effects of introducing WGN at the output of different convolutional layers on the accuracy of the CNN

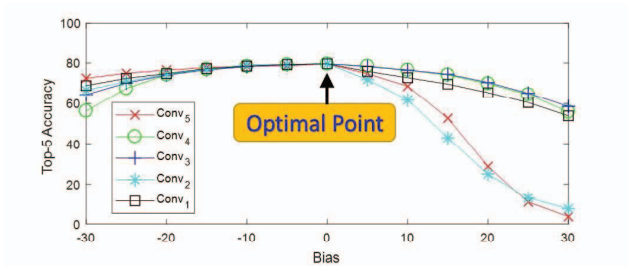

Fig. 6: Effects of introducing bias + 20dBW WGN on the accuracy of the CNN

V. Case study for significance-driven classification of filters

Multiple techniques are available in literature for pruning NNs. However, structured pruning, specifically in CNNs, have been reported to produce higher computational efficiency as compared to unstructured pruning [13]. The structured pruning techniques include, filter/channel pruning and intra-kernel strided sparsity techniques. Moreover, the intra-kernel strided sparsity has been reported to produce significant performance gains only when combined with convolutional lowering [13]. However, the filter/channel pruning directly reduces the required number of computations without any implications. In addition to this, as stated in section II, each filter in i^{th} convolutional layer requires $h_{i+1} \times w_{i+1} \times k_i^2 \times x_i$ computations to produce a single channel of the output feature map fm_{i+1}. Pruning a complete filter not only reduces the stated number of computations but also results in the removal of corresponding channels of the filters from the succeeding convolutional/fully-connected layer, which further reduces the number of computations by $w_{i+2} \times x_{i+2} \times k_{i+1}^2 \times h_{i+2}$.

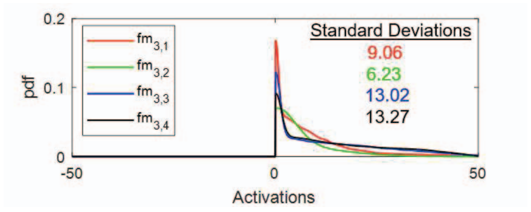

Fig. 7: Data distribution of 4 different channels of fm_3 of VGG-f network

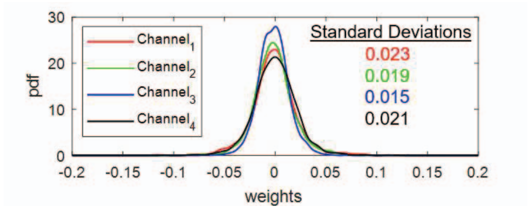

Fig. 8: Distribution of weights in 4 different channels of the filters of 3^{rd} convolutional layer

State-of-the-art techniques for filter/channel pruning use L1-/L2-norm of filters for quantifying their significance within a convolutional layer [6]. However, we propose to use a data aware technique that computes the significance $'s'$ of a j^{th} filter within an i^{th} CONV layer based upon the distribution of it's contribution to the output of $i+1^{th}$ layer. Although the contribution of a j^{th} filter within an i^{th} CONV layer to the output of $i+1^{th}$ layer should be computed using combined distribution of the convolutions of the corresponding feature map $fm_{i+1,j}$ with j^{th} channels of all the filters in CONV layer $i+1$ (i.e., $f_{i+1,1,j}, f_{i+1,2,j}, ..., f_{i+1,N_{i+1},j}$), we estimate the contribution using the combined distribution of the products of the entities in feature map $fm_{i+1,j}$ and the weights in respective j^{th} channels of all the filters of layer $i+1$ (i.e., $f_{i+1,1,j}, f_{i+1,2,j}, ..., f_{i+1,N_{(i+1)},j}$). This provides us with almost the same relative measures as in the former case due to the fact the the basic operations (for computing a single output value) in convolution are element wise multiplications followed by additions. In this work, we estimate the distribution of the products using distribution of feature maps and the respective filter channels as illustrated in fig. 9. Few realistic data and weight distributions are illustrated in fig. 7 and 8, respectively. Note that here fm_{i+1} corresponds to the feature maps exactly at the input of $i+1^{th}$ layer, i.e., after passing through all the intermediate pooling, activation, and/or normalization layers between i^{th} and $i+1^{th}$ computational layers.

The detailed steps for estimating the significance $'s'$ of filters of an i^{th} convolutional layer are as follows:

1) Simulate the CNN architecture for a small subset of input samples and compute the standard deviation and mean of input activations of $i+1^{th}$ layer, denoted by $\sigma(fm_{i+1,j})$ and $\mu(fm_{i+1,j})$, respectively.
2) Compute the standard deviation and mean of the weights of the corresponding channels of filters in $i+1^{th}$ CONV layer, denoted by $\sigma(f_{i+1,:,j})$ and $\mu(f_{i+1,:,j})$, respectively.
3) Using the standard deviation and mean of activations and weights, estimate the standard deviation and mean of $fm_{i+1,j} \times f_{i+1,:,j}$. Assuming activations and weights to be independent, we can compute the standard deviation and mean using following equations:

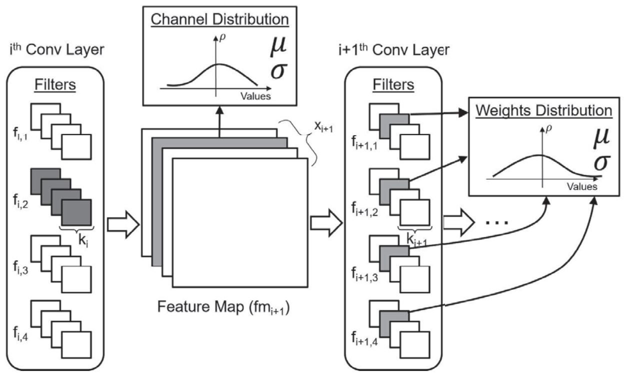

Fig. 9: Filter significance evaluation

$$\sigma_{i,j} = \sqrt{(\sigma(fm_{i+1,j})^2 \times \sigma(f_{i+1,:,j})^2 + \sigma(fm_{i+1,j})^2 \times \mu(f_{i+1,:,j})^2 + \sigma(f_{i+1,:,j})^2 \times \mu(fm_{i+1,j})^2)} \quad (1)$$

$$\mu_{i,j} = \mu(fm_{i+1,j}) \times \mu(f_{i+1,:,j}) \quad (2)$$

4) For each filter compute $s_j = 2 \times \sigma_{i,j} + \mu_{i,j}$
5) Sort the filters by their respective significance values s_j
6) For approximation, prune $'Y'$ number of filters with the smallest s_j

A. Comparison with State-of-the-art

To analyze the effectiveness of the proposed technique, we evaluated the accuracy of the network, for each layer independently, as the least significant filters were pruned away. We used VGG-f architecture [15] for classifying 5000 randomly selected images from the ImageNet validation set [17]. The testing set was kept the same throughout the results generation process. For computing the standard deviation and mean of the channels, for the proposed technique, we used 2000 randomly selected from the same ImageNet validation set. Fig. 10 shows the results when filters were sorted using L1-norm of filter weights while fig. 11 shows the results when the filters were sorted using the proposed technique.

The results in fig. 10 and 11 illustrates that for CONV layers 1, 2, and 4 the proposed technique provides better prediction while producing comparable results for rest of the CONV layers.

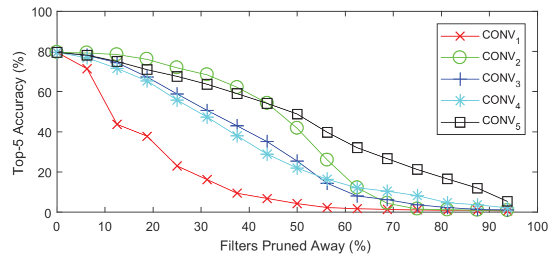

Fig. 10: Effects of significance-driven pruning of filters on the output accuracy using state-ofthe-art method [6]

VI. Conclusion

Error resilience analysis is vital for estimating the tolerable amount of approximations that can be employed for improving the performance efficiency of a Convolutional Neural Network (CNN). Towards this end, we proposed a methodology for employing approximations in CNNs. We also proposed a method for estimating the significance of convolutional layers for hardware approximations and for estimating the significance of filters within convolutional layers at software level.

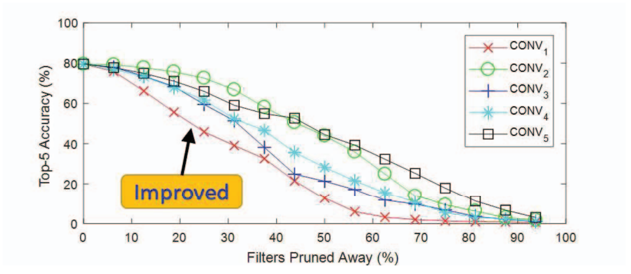

Fig. 11: Effects of significance-driven pruning of filters on the output accuracy using proposed method

References

[1] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in neural information processing systems*, 2012, pp. 1097–1105.
[2] F. Schroff, D. Kalenichenko, and J. Philbin, "Facenet: A unified embedding for face recognition and clustering," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 815–823.
[3] C. Szegedy, W. Liu, Y. Jia, P. Sermanet, S. Reed, D. Anguelov, D. Erhan, V. Vanhoucke, and A. Rabinovich, "Going deeper with convolutions," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2015, pp. 1–9.
[4] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.
[5] S. Zagoruyko and N. Komodakis, "Wide residual networks," *arXiv preprint arXiv:1605.07146*, 2016.
[6] H. Li, A. Kadav, I. Durdanovic, H. Samet, and H. P. Graf, "Pruning filters for efficient convnets," *arXiv preprint arXiv:1608.08710*, 2016.
[7] W. Chen, J. Wilson, S. Tyree, K. Weinberger, and Y. Chen, "Compressing neural networks with the hashing trick," in *International Conference on Machine Learning*, 2015, pp. 2285–2294.
[8] D. Lin, S. Talathi, and S. Annapureddy, "Fixed point quantization of deep convolutional networks," in *International Conference on Machine Learning*, 2016, pp. 2849–2858.
[9] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural networks with pruning, trained quantization and huffman coding," *arXiv preprint arXiv:1510.00149*, 2015.
[10] H. R. Mahdiani, A. Ahmadi, S. M. Fakhraie, and C. Lucas, "Bio-inspired imprecise computational blocks for efficient vlsi implementation of soft-computing applications," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 57, no. 4, pp. 850–862, April 2010.
[11] Z. Du, A. Lingamneni, Y. Chen, K. V. Palem, O. Temam, and C. Wu, "Leveraging the error resilience of neural networks for designing highly energy efficient accelerators," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 8, pp. 1223–1235, Aug 2015.
[12] H. Nakahara and T. Sasao, "A deep convolutional neural network based on nested residue number system," in *2015 25th International Conference on Field Programmable Logic and Applications (FPL)*, Sept 2015, pp. 1–6.
[13] S. Anwar, K. Hwang, and W. Sung, "Structured pruning of deep convolutional neural networks," *ACM Journal on Emerging Technologies in Computing Systems (JETC)*, vol. 13, no. 3, p. 32, 2017.
[14] M. Shafique, R. Hafiz, S. Rehman, W. El-Harouni, and J. Henkel, "Invited - cross-layer approximate computing: From logic to architectures," in *Proceedings of the 53rd Annual Design Automation Conference*, ser. DAC '16. New York, NY, USA: ACM, 2016, pp. 99:1–99:6. [Online]. Available: http://doi.acm.org/10.1145/2897937.2906199
[15] A. Vedaldi and K. Lenc, "Matconvnet – convolutional neural networks for matlab," in *Proceeding of the ACM Int. Conf. on Multimedia*, 2015.
[16] A. Leon-Garcia, *Probability and Random Processes For EE's (3rd Edition)*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 2007.
[17] 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*, vol. 115, no. 3, pp. 211–252, 2015.