Full Approximation of Deep Neural Networks through Efficient Optimization

Cecilia De la Parra
Robert Bosch GmbH
Renningen, Germany
cecilia.delaparra@de.bosch.com

Andre Guntoro
Robert Bosch GmbH
Renningen, Germany
andre.guntoro@de.bosch.com

Akash Kumar
Technological University of Dresden
Dresden, Germany
akash.kumar@tu-dresden.de

***Abstract*—Approximate Computing is a promising paradigm for mitigating computational requirements of Deep Neural Networks (DNN), by taking advantage of their inherent error resilience. Specifically, the use of approximate multipliers in DNN inference can lead to significant improvements in power consumption of embedded DNN applications. This paper presents a methodology for efficient approximate multiplier selection and for full and uniform approximation of large DNNs, through retraining and minimization of the approximation error. We evaluate our methodology using 422 approximate multipliers from the EvoApprox library, with three different Residual architectures trained with Cifar10, and achieve energy savings of up to 18% surpassing the original floating-point accuracy, and of up to 58% with an accuracy loss of 0.73%.**

I. Introduction

Through approximate computing, hardware requirements in error-resilient applications such as DNNs can be reduced. Approximations at software level are e.g. precision scaling or quantization [1], [2], [3]. At hardware level, approximation of arithmetic units such as multipliers [4] can lead to significant energy savings. In this context, DNNs can be partially or fully approximated. Techniques for partial approximation include retraining [4], [5], and more recently, authors in [6] proposed *ALWANN*, a framework for partial DNN approximation without retraining. Drawbacks of partial approximation are the energy savings, limited by the amount of approximated elements, and its challenging implementation in generic hardware accelerators. Furthermore, the search for optimal designs is highly time-consuming in traditional simulation frameworks [6]. On the other hand, through specialized methodologies for full approximation of DNNs, a more generic implementation and more energy savings can be achieved. In [7], authors investigate fully approximated and retrained DNNs with quantized weights and FP activations. Methods for optimizing the Approximate Multiplier (AM) itself were introduced in [8], [9], which proved to be effective but time-consuming and with low flexibility, and authors in [10], [11] demonstrated the viability of accuracy recovery through hardware-aware retraining in DNNs for digit recognition. Specialized simulation frameworks such as *Concrete* [12], based on Caffe [13], have been recently proposed to further accelerate approximate DNN computation in Graphic Processing Units (GPUs) with small time overhead, making the optimization of larger approximate DNNs more attractive.

Considering all currently available approximate hardware designs, selecting the appropiate AM for uniform DNN inference without accuracy loss is a non trivial task, which depends mainly on the availability of resources for DNN retraining. For some applications, such as those with proprietary DNNs, retraining with the original dataset is not possible as it is not available to the public, and therefore, other approaches to overcome this are needed. On the other hand, retraining with all possible design points given all existing AMs is a time exhaustive task even with specialized simulation frameworks. For example, we would need around 1.5 days just to train a ResNet20 [14] with every multiplier from [15] for only one epoch with our specialized simulation framework. With this motivation, we present a methodology for efficient multiplier selection and full approximation of large DNNs without accuracy loss. We focus on designs with the same multiplier for all convolutional and fully-connected (FC) layers, which is desirable for hardware accelerators with uniform processing elements. For this, given an initial set of AMs, we first select the Pareto set of power consumption and Mean Relative Error (MRE), as we prove that MRE is highly correlated to DNN accuracy. Through this, we restrict our search to 18% of the initial design space. Then, we retrain only the approximate DNNs with multipliers from this Pareto set. Our methodology allows retraining even without the original dataset, as we can switch to minimizing the propagated approximation error instead of optimizing the DNN accuracy itself.
The novel contributions presented in this work are:

- A data-driven analysis for multiplier selection based on MRE, focused on convolutional and FC layers.
- a methodology for efficient uniform and full approximation of DNNs.
- An alternative to DNN retraining when the original training dataset is not available.
- Significant power improvements without accuracy loss of three residual architectures [14] trained with Cifar10, through simulation of 422 AMs from the EvoApprox library [15].

II. Preliminaries

A. Deep Neural Networks

DNNs are characterized by the use of convolutional and FC layers. These operations can be generalized as follows: $y =$

978-1-7281-3320-1/20/$31.00 ©2020 IEEE

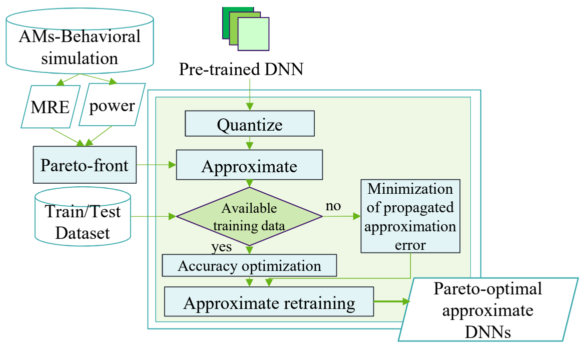

Fig. 1: Fast design exploration of approximate DNNs

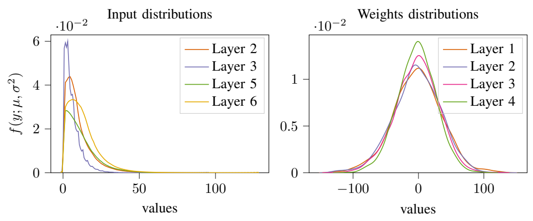

Fig. 2: Parameter distributions on ResNet8

$\Psi(g(x,w)+b)$, where y is the layer output, g is the weighted operation between inputs x and weights w, b is the layer bias, and Ψ is a non-linear activation function, usually a Rectified Linear Unit (ReLU) [16]. To reduce energy consumption, we quantize all weights and activations to int8 precision. In this work, we employ linear, layer-wise quantization [3].

B. Approximate Multipliers

We implement the behavioral model of 422 different 8-bit AMs from the EvoApprox library [15]. As reference for 100% energy consumption, we use the accurate multiplier with the lowest power consumption (mul8_433), and we discard all multipliers with higher power. We define $f(x,y)_{acc}$ as the accurate multiplication between inputs x and y. For an AM, the operation is defined as $f(x,y)_{approx} = x \times y + \epsilon$, where ϵ is the approximation error, and the corresponding error metrics are formally computed within this range for each possible value. To compare the approximation error of different AMs, we use the Mean Relative Error (MRE) as in (1),where $n = 2^{bw}-1$, and a maximum function is included to avoid division by 0 and to impose a larger error penalization when $f(i,j)_{acc} = 0$. The motivation of choosing the MRE is presented in detail in section III-A.

$$MRE = \frac{1}{n}\sum_{i=0}^{n}\sum_{j=0}^{n}\frac{|f(i,j)_{approx} - f(i,j)_{acc}|}{\max(1,|f(i,j)_{acc}|)} \quad (1)$$

III. Methodology

Our proposed methodology is presented in Fig. 1. All key steps are detailed in the following sub-sections.

A. Pareto-set generation

For a given set of AMs, their Pareto front of power consumption and DNN accuracy represents an optimal solution. However, to generate this set we would need to implement all possible design choices, which is highly time-consuming. To avoid this, we propose to generate a sub-optimal solution, by obtaining the Pareto front of power and an error metric highly correlated to the DNN accuracy, similar to [9], where a Weighted MRE (WMRE) is proposed. To this end, we propose to use the MRE, and to showcase its advantage against more specialized metrics like the WMRE, we perform the following analysis: We adapt the WMRE to generalize all DNN layers and activations. We focus on convolutional and FC layers, where most weights and inputs are characterized by Gaussian distributions. This is supported by recent research works [17], [18], and holds for our case studies, e.g. in Fig. 2. Note that this analysis can be extended to other data distributions present in different DNN layers. The WMRE is defined as:

$$WMRE = \frac{1}{n}\sum_{i=0}^{n}\sum_{j=0}^{n}|f(i,j)_{approx} - f(i,j)_{acc}|\, w_i \quad , \quad (2)$$

In [9], weights w_i are defined individually for each DNN layer, according to the corresponding weights distribution. To obtain weights which generalize to all DNN layers and also to activations distributions, we instead define our goal as follows: To give more relevance to more probable combinations of $x = i$ and $y = j$. Then, $w_{i,j}$ can be derived from the multivariate distribution $\mathcal{N}(x,y;\mu,\Sigma)$. We make the following generalizations:

- Weights have Gaussian distribution with mean $\mu = 0$ and std. deviation σ_x. If the weights mean is not 0, we add an offset to the DNN weights as follows: $\mu + \text{offset} = 0 \rightarrow \widetilde{\mu} = 0$ to perform this analysis.
- All post-ReLU activations have rectified Gaussian distribution with standard deviation σ_y and probability density functions as follows:
$$f(y;\mu,\sigma^2) = \Phi\left(-\frac{\mu}{\sigma}\right)\delta(y) + \frac{1}{\sqrt{2\pi\sigma^2}}e^{-\frac{(y-\mu)^2}{2\sigma^2}}U(y) \quad , \quad (3)$$
where $\Phi\left(-\frac{\mu}{\sigma}\right)$ is the cumulative distribution function of the normal distribution, $\delta(y)$ is a Dirac impulse and $U(y)$ is a unit step [19]. Thus, for $y > 0$, the distribution is Gaussian. For the case of $y = 0$, we implicitly integrate a formal guarantee of maximum error penalization.

For a parameter-invariant case, we approximate $\sigma_y \simeq \sigma_x = \sigma$. The resulting probability density function is:

$$f(x,y;\sigma) = \frac{1}{2\pi\sigma^2}\exp^{-\frac{x^2+y^2}{2\sigma^2}} \quad (4)$$

Eliminating constants outside the exponent, and substituting (4) in (2), we obtain:

$$WMRE = \frac{1}{n}\sum_{i=1}^{n}|z_{approx,i} - z_{acc,i}|\exp^{-\frac{\beta(x^2+y^2)}{2\sigma^2}} \quad , \quad (5)$$

Where $\beta = 0$ if $x \times y = 0$ and 1 otherwise. Although this introduces a strong non-linearity, this also guarantees a maximum error penalization when $x \times y = 0$.
For our case studies, the Pearson's correlation coefficient ρ of WMRE and MRE of all EvoApprox multipliers indicates a

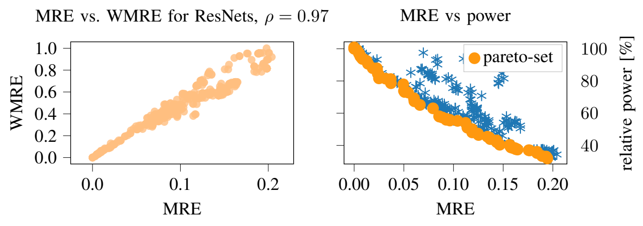

Fig. 3: MRE vs WMRE vs power of EvoApprox Multipliers

linear relationship (see Fig. 3). The ρ of WMRE and ResNet accuracies are between 0.74 and 0.85, indicating that WMRE as well as MRE are indeed affine towards evaluating DNN approximation errors. As the ρ of WMRE and Mean Absolute Error (MAE) or Root Mean Squared Error (RMSE) is very low (0.05 and 0.04 respectively), we conclude that MRE is the most suitable metric to generate our Pareto-front of AMs, shown in Fig. 3. The advantage of using MRE against WMRE is that we do not need the distribution values of the examined DNN for its computation.

B. Definition of loss function

After the given pre-trained DNN model is quantized and approximated, we retrain the DNN. For this, we first determine the loss function. In our case studies, we use the cross-entropy loss, defined as in (6), where y_k is the label, p_k the output, θ the DNN parameters and n the number of prediction classes.

$$C_\theta(p) = -\sum_{k=1}^{n} y_k \log p_k \tag{6}$$

Then, we determine the objective to be optimized. In our methodology, we have two possible optimization objectives:

- Accuracy optimization. If the original training dataset is available, we retrain with (7), where $\widetilde{p}_k$ is the approximate DNN output.

$$C_\theta(\widetilde{p}) = -\sum_{k=1}^{n} y_k \log \widetilde{p}_k \tag{7}$$

- Minimization of the propagated approximation error (MinPropAE). If the original training dataset is not available, we propose to generate an auxiliary dataset using the DNN with Floating-Point (FP) accuracy and without approximation. For this, we first generate a balanced dataset (same number of samples for each class) in the order of 100 to 1000 samples per class for a given DNN. Then, we input each data sample to the DNN and compute the corresponding FP-output p_k. This output will then be used as hard label of the corresponding input for retraining. Substituting in (7) we have:

$$C_\theta(\widetilde{p}_t) = -\sum_{k=1}^{n} p_k \log \widetilde{p}_k \quad , \tag{8}$$

where $\widetilde{p}_k$ is the approximate prediction at iteration t. p_k is a static parameter, independent of the updated weights. Thus, we are explicitly minimizing the propagated quantization error with respect to the FP accuracy.

Through approximate DNN retraining we impose soft constraints, as the use of weights that lead to approximation errors is not prohibited but has a well-defined penalization, and thus, this steers the weights to their error-free neighborhoods (determined by their MRE), if any, which mantains the proportionality between MRE and DNN accuracy after retraining.

C. Approximate retraining

The corresponding training loss is hereby minimized through stochastic gradient descent (SGD). The gradient update is computed using Straight-Through-Estimators (STE) as in [11], except for the quantization function:

$$w_{t+1} = w_t + \Delta w = w_t - \nabla \frac{\partial C(\widetilde{p})}{\partial w} \quad \text{, where:} \tag{9}$$

$$\frac{\partial C(\widetilde{p})}{\partial w} = \frac{\partial C(\widetilde{p})}{\partial \widetilde{p}} \underbrace{\frac{\partial p}{\partial W_q}}_{STE} \frac{\partial W_q}{\partial w} \quad , \tag{10}$$

where W_q are the quantized weights. In the STE the gradient is computed with respect to the accurate operation, as the derivative of approximated operands is undefined. Through this, we compensate simultaneously the quantzation and the approximation error.

IV. Evaluation

A. Simulation framework

For analysis and evaluation we use ProxSim [20] based on CUDA [21], a library for GPU parallel programming, and Tensorflow [22]. For efficient computation of approximate DNNs, we perform dynamic allocation in the GPU to load the behavioral simulation of the corresponding AM. All experiments were performed with an Nvidia GeForce GTX 1080Ti. We implement three residual architectures [14] initially trained with Cifar-10, during 200 epochs with an initial learning rate (lr) of 1e-3, with FP precision and without approximations. The characteristics of these DNNs are described in Table I.

TABLE I: Characteristics of implemented ResNets

DNN	FP Acc.	8b Acc.	#Params.	#MAC	Val.time
ResNet8	85.68%	84.61%	78,666	12.5M	12.9 s
ResNet14	89.41%	89.20%	176,554	26.7M	25.7 s
ResNet20	91.04%	90.34%	274,442	40.8M	37.9 s

B. Overall results

We test all multipliers from Section II-B in all DNNs from Table I. As in [6], we propose an accuracy tolerance of 1% with respect to the 8b accuracy. The results are plotted in Fig. 4. Note that we only plot the multipliers with accuracy above 80% for clear visualization with respect to the accuracy tolerance, and that the FP accuracy is never surpassed. The Pareto set of MRE vs. power has a mean distance of 5% to the Pareto-front of DNN accuracy vs. power, and thus presents a suboptimal solution of the design space. This Pareto set contains 75 AMs, which is only 18% of the original set. In this work, we keep the whole Pareto set for retraining to validate our approach, but after performing DNN inference

TABLE II: Comparison with ALWANN

DNN	ours						ALWANN				
	#AMs	Train/epoch time	Total time	Energy savings	Acc. [%]	Acc. loss [%]	#AMs	Total time	Energy savings	Acc. [%]	Acc. loss [%]
ResNet8		92 sec.	**1.91 hrs**	**58 %**	**83.88**	**0.73**		0.7 days	30 %	81.56	1.7
ResNet14	75	182 sec.	**3.79 hrs**	**57 %**	**88.34**	**0.86**	36	2.3 days	30 %	84.65	0.9
ResNet20		260 sec.	**5.42 hrs**	**53 %**		**0.74**		–	89.6	–	–

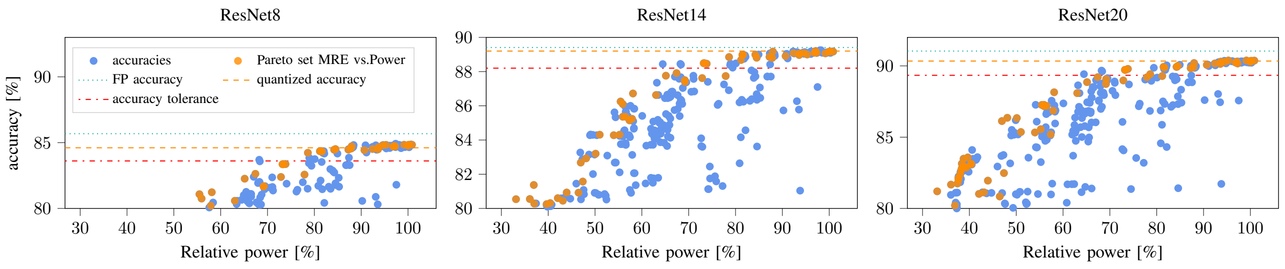

Fig. 4: Power vs accuracies of EvoApprox multipliers before retraining

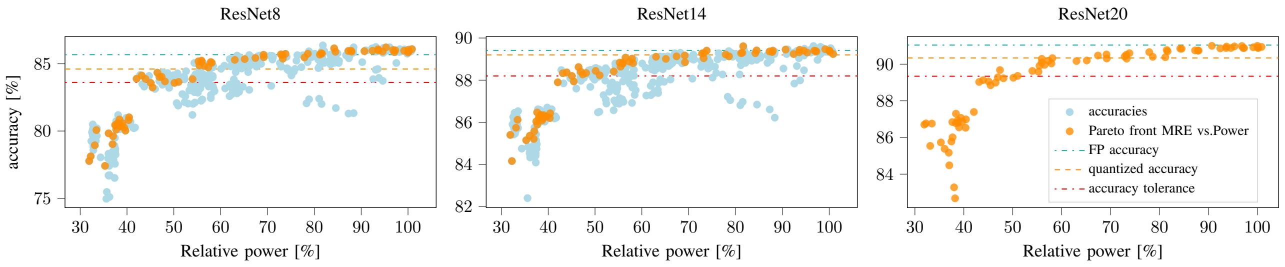

Fig. 5: Power vs accuracies of EvoApprox multipliers after retraining with accuracy optimization

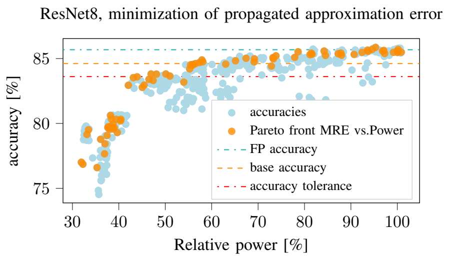

Fig. 6: Power vs accuracies after retraining with MinPropAE

on the Pareto set, it is recommended to retrain only with the multipliers below our accuracy tolerance. With this, we further reduce our design space to around 50 multipliers (12%) depending on the DNN.
For ResNet8 and ResNet14, we perform approximate retraining with accuracy optimization for one epoch, for all AMs, to corroborate that our selected Pareto set is still suboptimal after DNN optimization. We use SGD with momentum, lr of 1e-3 and batch size of 512. We find that one epoch is enough to compensate for at least 85% of the lost accuracy due to approximations, making our proposal for approximate retraining less time-consuming than other approaches. As we demonstrate that the initial Pareto front is also suboptimal after retraining, we retrain ResNet20 only with the AMs of the Pareto front. The results are plotted in Fig. 5. With our method, the original FP accuracy can be reached or even surpassed in all DNNs with different multipliers. With a better accuracy compared to the original FP accuracy, we reach energy savings of up to 15%, 18% and 9% in the case of ResNet8, ResNet14 and ResNet20 respectively.
We compare retraining with accuracy optimization and MinPropAE by retraining ResNet8 with the latter approach. The results are presented in Fig. 6. We observe that this method is also effective to recover DNN accuracy and thus is a valid alternative when the original dataset is not available. However, the FP accuracy can not be surpassed.
The comparison of our work with ALWANN [6], with respect to DNN accuracy and retraining/excecution times with approximate multipliers, is presented in Table II. With this results, we demonstrate that retraining with arbitrary AMs is feasible for large approximate DNNs given the proper methodology and simulation tools, contrary to the stated in [6].

V. Conclusions

In this paper, we propose a novel methodology for efficient multiplier selection and full and uniform approximation of DNNs through retraining, for reducing the energy consumption of multiplications in convolutional and FC layers. Additionally, we present an alternative method for retraining without the original training dataset, specially helpful when approximating proprietary DNNs. Although we are limited by not considering partial approximation, we demonstrate that through effective DNN retraining and specialized tools, we achieve better energy savings in shorter execution times, compared to state-of-the-art frameworks for partial DNN approximation.

References

[1] S. Gupta, A. Agrawal, K. Gopalakrishnan, and P. Narayanan, "Deep learning with limited numerical precision," *ICML '15*.

[2] S. Han *et al.*, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," *ICLR '16*.

[3] S. R. Jain, A. Gural, M. Wu, and C. Dick, "Trained uniform quantization for accurate and efficient neural network inference on fixed-point hardware," *CoRR*, vol. abs/1903.08066, 2019. [Online]. Available: http://arxiv.org/abs/1903.08066

[4] S. Venkataramani *et al.*, "AxNN: Energy-efficient neuromorphic systems using approximate computing," *ISLPED '14*.

[5] Q. Zhang *et al.*, "ApproxANN: An approximate computing framework for artificial neural network," in *DATE '15*.

[6] V. Mrazek, Z. Vasícek, L. Sekanina, M. A. Hanif, and M. Shafique, "ALWANN: automatic layer-wise approximation of deep neural network accelerators without retraining," *ICCAD '19*.

[7] S. S. Sarwar, S. Venkataramani, A. Ankit, A. Raghunathan, and K. Roy, "Energy-efficient neural computing with approximate multipliers," *J. Emerg. Technol. Comput. Syst.*, 2018.

[8] V. Mrazek, S. S. Sarwar, L. Sekanina, Z. Vasicek, and K. Roy, "Design of power-efficient approximate multipliers for approximate artificial neural networks," in *ICCAD '16*.

[9] Z. Vasicek, V. Mrazek, and L. Sekanina, "Automated Circuit Approximation Method Driven by Data Distribution," *DATE '19*.

[10] X. He, L. Ke, W. Lu, G. Yan, and X. Zhang, "Axtrain: Hardware-oriented neural network training for approximate inference," *ISLPED '18*.

[11] Y. Fan, X. Wu, J. Dong, and Z. Qi, "Axdnn: Towards the cross-layer design of approximate dnns," in *ASPDAC '19*.

[12] Z. Liu, G. Li, F. Qiao, Q. Wei, P. Jin, X. Liu, and H. Yang, "Concrete: A per-layer configurable framework for evaluating dnn with approximate operators," in *ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, May 2019, pp. 1552–1556.

[13] Y. Jia, E. Shelhamer, J. Donahue, S. Karayev, J. Long, R. Girshick, S. Guadarrama, and T. Darrell, "Caffe: Convolutional architecture for fast feature embedding," *arXiv preprint arXiv:1408.5093*, 2014.

[14] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *CVPR '15*.

[15] V. Mrazek, R. Hrbacek, Z. Vasicek, and L. Sekanina, "Evoapprox8b: Library of approximate adders and multipliers for circuit design and benchmarking of approximation methods," in *DATE '17*.

[16] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, ch. Deep feedforward networks.

[17] S. Han, H. Mao, and W. J. Dally, "Deep compression: Compressing deep neural network with pruning, trained quantization and huffman coding," in *ICLR '16*.

[18] E. Park, J. Ahn, and S. Yoo, "Weighted-entropy-based quantization for deep neural networks," in *CVPR '17*.

[19] M. Harva and A. Kabán, "Variational learning for rectified factor analysis," *Signal Process.*, 2007.

[20] C. De la Parra, A. Guntoro, and A. Kumar, "Proxsim: Gpu-based simulation framework for cross-layer approximate dnn optimization," in *DATE '20*.

[21] J. Nickolls, I. Buck, M. Garland, and K. Skadron, "Scalable parallel programming with cuda," *Queue*, 2008.

[22] M. Abadi, A. Agarwal, P. Barham, E. Brevdo, Z. Chen, C. Citro *et al.*, "TensorFlow: Large-scale machine learning on heterogeneous systems," 2015, software available from tensorflow.org. [Online]. Available: https://www.tensorflow.org/