Accepted as a conference paper at ACML 2017

NeuralPower: Predict and Deploy Energy-Efficient Convolutional Neural Networks

Ermao Cai ERMAO@CMU.EDU
Department of ECE, Carnegie Mellon University, Pittsburgh, PA, USA

Da-Cheng Juan DACHENG@GOOGLE.COM
Google Research, Mountain View, CA, USA

Dimitrios Stamoulis DSTAMOUL@ANDREW.CMU.EDU
Department of ECE, Carnegie Mellon University, Pittsburgh, PA, USA

Diana Marculescu DIANAM@CMU.EDU
Department of ECE, Carnegie Mellon University, Pittsburgh, PA, USA

Editors: Yung-Kyun Noh and Min-Ling Zhang

arXiv:1710.05420v1 [cs.LG] 15 Oct 2017

Abstract

"How much energy is consumed for an inference made by a convolutional neural network (CNN)?" With the increased popularity of CNNs deployed on the wide-spectrum of platforms (from mobile devices to workstations), the answer to this question has drawn significant attention. From lengthening battery life of mobile devices to reducing the energy bill of a datacenter, it is important to understand the energy efficiency of CNNs during serving for making an inference, before actually training the model. In this work, we propose *NeuralPower*: a layer-wise predictive framework based on sparse polynomial regression, for predicting the serving energy consumption of a CNN deployed on any GPU platform. Given the architecture of a CNN, *NeuralPower* provides an accurate prediction and breakdown for power and runtime across all layers in the whole network, helping machine learners quickly identify the power, runtime, or energy bottlenecks. We also propose the *"energy-precision ratio" (EPR)* metric to guide machine learners in selecting an energy-efficient CNN architecture that better trades off the energy consumption and prediction accuracy. The experimental results show that the prediction accuracy of the proposed *NeuralPower* outperforms the best published model to date, yielding an improvement in accuracy of up to 68.5%. We also assess the accuracy of predictions at the network level, by predicting the runtime, power, and energy of state-of-the-art CNN architectures, achieving an average accuracy of 88.24% in runtime, 88.34% in power, and 97.21% in energy. We comprehensively corroborate the effectiveness of *NeuralPower* as a powerful framework for machine learners by testing it on different GPU platforms and Deep Learning software tools.

Keywords: Power, Runtime, Energy, Polynomial Regression, Convolutional Neural Networks.

1. Introduction

"How much energy is consumed for an inference made by a convolutional neural network (CNN)?" "Is it possible to predict this energy consumption before a model is even trained?" "If yes, how should machine learners select an energy-efficient CNN for deployment?" These three questions serve as the main motivation of this work.

In recent years, CNNs have been widely applied in several important areas, such as text processing and computer vision, in both academia and industry. However, the high energy consumption of CNNs has limited the types of platforms that CNNs can be deployed on, which can be attributed to both (a) high power consumption and (b) long runtime. GPUs have been adopted for performing CNN-related services in various computation environments ranging from data centers, desktops,

© E. Cai, D.-C. Juan, D. Stamoulis & D. Marculescu.

to mobile devices. In this context, resource constraints in GPU platforms need to be considered carefully before running CNN-related applications.

In this paper, we focus on the *testing* or *service* phase since, CNNs are typically deployed to provide services (*e.g.*, image recognition) that can potentially be invoked billions of times on millions of devices using the same architecture. Therefore, testing runtime and energy are critical to both users and cloud service providers. In contrast, training a CNN is usually done once. Orthogonal to many methods utilizing hardware characteristics to reduce energy consumptions, CNN architecture optimization in the design phase is significant. In fact, given the same performance level (*e.g.*, the prediction accuracy in image recognition task), there are usually many CNNs with different energy consumptions. Figure 1 shows the relationship between model testing errors and energy consumption for a variety of CNN architectures. We observe that several architectures can achieve a similar accuracy level. However, the energy consumption drastically differs among these architectures, with the difference as large as 40× in several cases. Therefore, seeking for energy-efficient CNN architecture without compromising performance seems intriguing, especially for large-scale deployment.

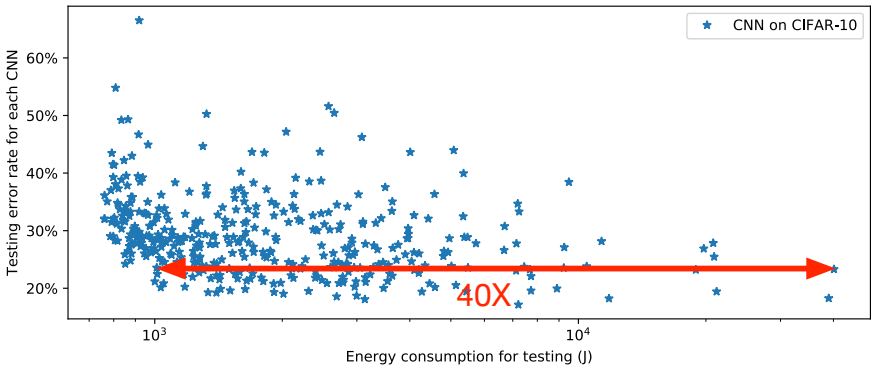

Figure 1: Testing error vs. energy consumption for various CNN architectures on CIFAR-10 running with TensorFlow on Nvidia Titan X GPU. Each point represents one randomly-sampled CNN architecture for making inferences on CIFAR-10. For the architectures that achieve similar error rates (around 20%) during test phase, the energy consumption can vary significantly by more than 40×.

To identify energy-efficient CNNs, the use of accurate runtime, power, and energy models is of crucial importance. The reason for this is twofold. First, commonly used metrics characterizing CNN complexity (*e.g.*, total FLOPs[1]) are too crude to predict energy consumption for real platforms. Energy consumption depends not only on the CNN architecture, but also on the software/hardware platform. In addition, it is also hard to predict the corresponding runtime and power. Second, traditional profiling methods have limited effectiveness in identifying energy-efficient CNNs, due to several reasons: 1) These methods tend to be inefficient when the search space is large (*e.g.*, more than 50 architectures); 2) They fail to quantitatively capture how changes in the CNN architectures affect runtime, power, and energy. Such results are critical in many automatic neural architecture search algorithms Zoph and Le (2016); 3) Typical CNNs are inconvenient or infeasible to profile if the service platform is different than the training platform. Therefore, it is imperative to train models for power, runtime, energy consumption of CNNs. Such models would significantly help Machine Learning practitioners and developers to design accurate, fast, and energy-efficient CNNs,

1. FLOP stands for "floating point operation".

especially in the design space of mobile or embedded platforms, where power- and energy-related issues are further exacerbated.

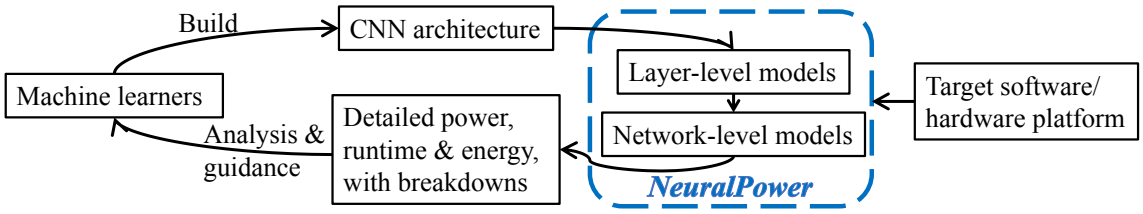

Figure 2: *NeuralPower* quickly predicts the power, runtime, and energy consumption of a CNN architecture during service phase. Therefore, *NeuralPower* provides the machine learners with analysis and guidance when searching for energy-efficient CNN architectures on given software/hardware platforms.

In this paper, we develop a predictive framework for power, runtime, and energy of CNNs during the testing phase, namely *NeuralPower*, *without* actually running (or implementing) these CNNs on a target platform. The framework is shown in Figure 2. That is, given (a) a CNN architecture of interest and (b) the target platform where the CNN model will be deployed, *NeuralPower* can directly predict the power, runtime, and energy consumption of the network in service/deployment phase. This paper brings the following contributions:

- To the best of our knowledge, our proposed learning-based polynomial regression approach, namely *NeuralPower*, is the first framework for predicting the *power consumption* of CNNs running on GPUs, with an average accuracy of 88.34%.
- *NeuralPower* can also predict *runtime* of CNNs, which outperforms state-of-the-art analytical models, by achieving an improvement in accuracy up to 68.5% compared to the best previously published work.
- *NeuralPower* uses power and runtime predictions to predict the *energy consumption* of CNNs running on GPUs, with an average accuracy of 97.21%.
- In addition to total runtime and average power, *NeuralPower* also provides the detailed breakdown of runtime and power across different components (at each layer) of the whole network, thereby helping machine learners to identify efficiently the runtime, power, or energy bottlenecks.
- We propose the *Energy-Precision Ratio* metric for guiding machine learners to trade-off between CNN classification accuracy and energy efficiency.

The rest of this paper is organized as follows. Section 2 introduces the background knowledge and related work. Section 3 provides the details of the proposed *NeuralPower*: layer-level and whole-network models, and the process (configurations and platforms) for collecting the dataset. Section 4 shows the experimental results on power, runtime, and energy predictions. Finally, Section 5 concludes the paper.

2. Background and Related Work

Prior art, *e.g.*, by Han et al. (2015), has identified the runtime overhead and power consumption of CNNs execution to be a significant concern when designing accurate, yet power- and energy-efficient CNN models. These design constraints become increasingly challenging to address, especially in

the context of two emerging design paradigms, *(i)* the design of larger, power-consuming CNN architectures, and *(ii)* the design of energy-aware CNNs for mobile platforms. Hence, a significant body of state-of-the-art approaches aim to tackle such runtime and power overhead, by accelerating the execution of CNNs and/or by reducing their power-energy consumption.

Runtime-efficient CNNs: To enable CNN architectures that execute faster, existing work has investigated both hardware- and software-based methodologies. On one hand, with respect to hardware, several hardware platforms have been explored as means to accelerate the CNN execution. Examples of these platforms include FPGA-based methodologies by Zhang et al. (2015a), ASIC-like designs by Zhang et al. (2015b). On the other hand, with respect to software-based acceleration, several libraries, such as cuDNN and Intel MKL, have been used in various deep learning frameworks to enable fast execution of CNNs, since these libraries provide specially-optimized primitives in CNNs to improve their runtime. Among the different hardware and software platforms, GPU-based frameworks are widely adopted both in academia and industry, thanks to the good trade-off between platform flexibility and performance that they provide. To this end, in this work, we aim to model the runtime and power characteristic of CNNs executing on state-of-the-art GPU platforms.

Power- and energy-aware CNNs: Recent work has focused on limiting the energy and power consumption of CNNs. Several approaches investigate the problem in the context of hyper-parameter optimization. For instance, Rouhani et al. (2016a) have proposed an automated customization methodology that adaptively conforms the CNN architectures to the underlying hardware characteristics, while minimally affecting the inference accuracy. Smithsons et al. (2016) have used a pair of trainer-trainee CNNs to incorporate accuracy and cost into the design of neural networks. Beyond these methods which are based on hyper-parameter optimization, another group of novel approaches focuses solely on the energy-efficient CNN implementation assuming a given network architecture. These approaches include techniques that draw ideas from energy-aware computer system design, such as the methodologies by Han et al. (2015) and Courbariaux et al. (2016).

Modeling of metrics characterizing CNNs: While all the aforementioned approaches motivate hardware-related constraints as limiting factors towards enabling efficient CNN architecture, to the best of our knowledge there is no comprehensive methodology that models the runtime, power, and eventually the energy of CNN architectures. That is, prior work either relies on proxies of memory consumption or runtime expressed as simplistic counts of the network weights (*e.g.*, as done by Rouhani et al. (2016b)), or extrapolates power-energy values from energy-per-operation tables (as assumed by Han et al. (2015)). Consequently, existing modeling assumptions are either overly simplifying or they rely on outdated technology nodes. Our work successfully addresses these limitations, by proposing power, energy, and runtime models that are validated against state-of-the-art GPUs and Deep Learning software tools.

A work that shares similar insight with our methodology is the *Paleo* framework proposed by Qi et al. (2016). In their approach, the authors present an analytical method to determine the runtime of CNNs executing on various platforms. However, their model cannot be flexibly used across different platforms with different optimization libraries, without detailed knowledge of them. More importantly, their approach cannot predict power and energy consumption.

3. Methodology: Power and Runtime Modeling

In this section, we introduce our hierarchical power and runtime model framework -*NeuralPower*- and the data collection process. *NeuralPower* is based on the following key insight: despite the huge amount of different CNN variations that have been used in several applications, all these CNN architectures consist of basic underlying building blocks/primitives which exhibit similar execution characteristics per type of layer. To this end, *NeuralPower* first models the power and runtime of the key layers that are commonly used in a CNN. Then, *NeuralPower* uses these models to predict the performance and runtime of the entire network.

3.1. Layer-Level Power and Runtime Modeling

The first part of *NeuralPower* is layer-level power and runtime models. We construct these models for each type of layer for both runtime and power. More specifically, we select to model three types of layers, namely the *convolutional*, the *fully connected*, and the *pooling* layer, since these layers carry the main computation load during CNN execution – as also motivated by Qi et al. (2016). Nevertheless, unlike prior work, our goal is to make our model flexible for various combinations of software/hardware platforms without knowing the details of these platforms.

To this end, we propose a learning-based *polynomial regression model* to learn the coefficients for different layers, and we assess the accuracy of our approach against power and runtime measurements on different commercial GPUs and Deep Learning software tools. There are three major reasons for this choice. *First*, in terms of model accuracy, polynomial models provide more flexibility and low prediction error when modeling both power and runtime. The *second* reason is the interpretability: runtime and power have clear physical correlation with the layer's configuration parameters (*e.g.*, batch size, kernel size, *etc.*). That is, the features of the model can provide an intuitive encapsulation of how the layer parameters affect the runtime and power. The *third* reason is the available amount of sampling data points. Polynomial models allow for adjustable model complexity by tuning the degree of the polynomial, ranging from linear model to polynomials of high degree, whereas a formulation with larger model capacity may be prone to overfitting. To perform model selection, we apply ten-fold cross-validation and we use Lasso to decrease the total number of polynomial terms. The detailed model selection process will be discussed in Section 4.1.

Layer-level runtime model: The runtime $\hat{T}$ of a layer can be expressed as:

$$\hat{T}(\mathbf{x}_T) = \sum_j c_j \cdot \prod_{i=1}^{D_T} x_i^{q_{ij}} + \sum_s c'_s \mathcal{F}_s(\mathbf{x}_T) \tag{1}$$

$$\text{where } \mathbf{x}_T \in \mathbb{R}^{D_T};\ q_{ij} \in \mathbb{N};\ \forall j,\ \sum_{i=1}^{D_T} q_{ij} \leq K_T.$$

The model consists of two components. The first component corresponds to the regular degree-K_T polynomial terms which are a function of the features in input vector $\mathbf{x}_T \in \mathbb{R}^{D_T}$. x_i is the i-th component of $\mathbf{x}_T$. q_{ij} is the exponent for x_i in the j-th polynomial term, and c_j is the coefficient to learn. This feature vector of dimension D_T includes layer configuration hyper-parameters, such as the batch size, the input size, and the output size. For different types of layers, the dimension D_T is expected to vary. For convolutional layers, for example, the input vector includes the kernel shape, the stride size, and the padding size, whereas such features are not relevant to the formulation/configuration of a fully-connected layer.

The second component corresponds to special polynomial terms $\mathcal{F}$, which encapsulate physical operations related to each layer (*e.g.*, the total number of memory accesses and the total number of floating point operations). The number of the special terms differs from one layer type to another. For the convolutional layer, for example, the special polynomial terms include the memory access count for input tensor, output tensor, kernel tensor, and the total number of floating point operations for the all convolution computations. Finally, c'_s is the coefficient of the s-th special term to learn.

Based on this formulation, it is important to note that not all input parameters are positively correlated with the runtime. For example, if the stride size increases, the total runtime will decrease since the total number of convolutional operations will decrease. This observation motivates further the use of a polynomial formulation, since it can capture such trends (unlike a posynomial model, for instance).

Layer-level power model: To predict the power consumption $\hat{P}$ for each layer type during *testing*, we follow a similar polynomial-based approach:

$$\hat{P}(\mathbf{x}_P) = \sum_j z_j \cdot \prod_{i=1}^{D_P} x_i^{m_{ij}} + \sum_k z'_k \mathcal{F}_k(\mathbf{x}_P) \tag{2}$$

$$\text{where } \mathbf{x}_P \in \mathbb{R}^{D_P};\ m_{ij} \in \mathbb{N};\ \forall j,\ \sum_{i=1}^{D_P} m_{ij} \leq K_P.$$

where the regular polynomial terms have degree K_P and they are a function of the input vector $\mathbf{x}_P \in \mathbb{R}^{D_P}$. m_{ij} is the exponent for x_i of the j-th polynomial term, and z_j is the coefficient to learn. In the second sum, z'_k is the coefficient of the k-th special term to learn.

Power consumption however has a non-trivial correlation with the input parameters. More specifically, as a metric, power consumption has inherent limits, *i.e.*, it can only take a range of possible values constrained through the power budget. That is, when the computing load increases, power does not increase in a linear fashion. To capture this trend, we select an extended feature vector $\mathbf{x}_P \in \mathbb{R}^{D_P}$ for our power model, where we include the logarithmic terms of the features used for runtime (*e.g.*, batch size, input size, output size, *etc.*). As expected, the dimension D_P is twice the size of the input feature dimension D_T. A logarithmic scale in our features vector can successfully reflect such a trend, as supported by our experimental results in Section 4.

3.2. Network-Level Power, Runtime, and Energy Modeling

We discuss the network-level models for *NeuralPower*. For the majority of CNN architectures readily available in a Deep Learning models "zoo" (as the one compiled by Jia et al. (2014)), the whole structure consists of and can be divided into several layers in series. Consequently, using our predictions for power and runtime as building blocks, we extend our predictive models to capture the runtime, the power, and eventually the energy, of the entire architecture at the *network level*.

Network-level runtime model: Given a network with N layers connected in series, the predicted total runtime can be written as the sum of the predicted runtime $\hat{T}_n$ of each layer n:

$$\hat{T}_{total} = \sum_{n=1}^{N} \hat{T}_n \tag{3}$$

Network-level power model: Unlike the summation for total runtime, the average power can be obtained using both per layer runtime and power. More specifically, we can represent the average power $\hat{P}_{avg}$ of a CNN as:

$$\hat{P}_{avg} = \frac{\sum_{n=1}^{N} \hat{P}_n \cdot \hat{T}_n}{\sum_{n=1}^{N} \hat{T}_n} \tag{4}$$

Network-level energy model: From here, it is easy to derive our model for the total energy consumption $\hat{E}_{total}$ of an entire network configuration:

$$\hat{E}_{total} = \hat{T}_{total} \cdot \hat{P}_{avg} = \sum_{n=1}^{N} \hat{P}_n \cdot \hat{T}_n \tag{5}$$

which is basically the scalar product of the layer-wise power and runtime vectors, or the sum of energy consumption for all layers in the model.

3.3. Dataset Collection

Experiment setup: The main modeling and evaluation steps are performed on the platform described in Table 1. To exclude the impact of voltage/frequency changing on the power and runtime data we collected, we keep the GPU in a fixed state and CUDA libraries ready to use by enabling the persistence mode. We use `nvidia-smi` to collect the instantaneous power per 1 ms for the entire measuring period. Please note that while this experimental setup constitutes our configuration basis for investigating the proposed modeling methodologies, in Section 4.4 we present results of our approach on other GPU platforms, such as Nvidia GTX 1070, and Deep Learning tools, such as Caffe by Jia et al. (2014).

Table 1: Target platform

CPU / Main memory	Intel Core-i7 5820K / 32GB
GPU	Nvidia GeForce GTX Titan X (12GB DDR5)
GPU max / idle power	250W / 15W
Deep learning platform	TensorFlow 1.0 on Ubuntu 14
Power meter	NVIDIA System Management Interface

CNN architectures investigated: To comprehensively assess the effectiveness of our modeling methodology, we consider several CNN models. Our analysis includes state-of-the-art configurations, such as the AlexNet by Krizhevsky (2014), VGG-16 & VGG-19 by Simonyan and Zisserman (2014), R-CNN by Ren et al. (2015), NIN network by Lin et al. (2013), CaffeNet by Jia et al. (2014), GoogleNet by Szegedy et al. (2015), and Overfeat by Sermanet et al. (2013). We also consider different flavors of smaller networks such as vanilla CNNs used on the MNIST by LeCun et al. (1998) and CIFAR10-6conv Courbariaux et al. (2015) on CIFAR-10. This way, we can cover a wider spectrum of CNN applications.

Data collection for power/runtime models: To train the layer-level predictive models, we collect data points by profiling power and runtime from all layers of all the considered CNN architectures in the training set. We separate the training data points into groups based on their layer types. In this paper, the training data include 858 convolution layer samples, 216 pooling layer samples, and 116 fully connected layer samples. The statistics can change if one needs any form of customization. For testing, we apply our learned models on the network in the testing set, and compare our predicted results against the actual results profiled on the same platform, including both layer-level evaluation and network-level evaluation.

4. Experimental Results

In this section, we assess our proposed *NeuralPower* in terms of power, runtime, and energy prediction accuracy at both layer level and network level. Since the models for runtime and power are slightly different from one to another, we discuss them separately in each case. We then propose a metric called *Energy-Precision Ratio* to guide machine learners toward energy-efficient CNNs. In the end of this section, we validate our framework on other hardware and software platforms.

4.1. Layer-Level Model Evaluation

4.1.1. Model selection

To begin with model evaluation, we first illustrate how model selection has been employed in *NeuralPower*. In general, *NeuralPower* changes the order of the polynomial (*e.g.*, D_T in Equation 1) to expand/shrink the size of feature space. *NeuralPower* applies Lasso to select the best model for each polynomial model. Finally, *NeuralPower* selects the final model with the lowest cross-validation Root-Mean-Square-Error (RMSE), which is shown in Figure 3.

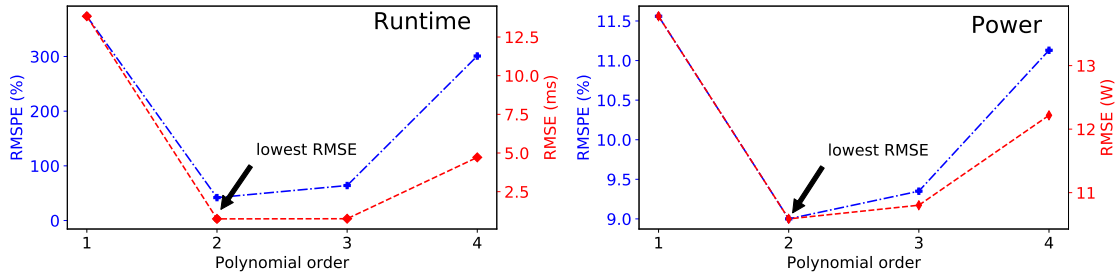

Figure 3: Comparison of best-performance model with respect to each polynomial order for the fully-connected layers. In this example, a polynomial order of two is chosen since it achieves the best Root-Mean-Square-Error (RMSE) for both runtime and power modeling. At the same time, it also has the lowest Root-Mean-Square-Percentage-Error (RMSPE).

4.1.2. Runtime Models

Applying the model selection process, we achieve a polynomial model for each layer type in a CNN. The evaluation of our models is shown in Table 2, where we report the Root-Mean-Square-Error (RMSE) and the relative Root-Mean-Square-Percentage-Error (RMSPE) of our runtime predictions for each one of the considered layers. Since we used Lasso in our model selection process, we also report the model size (*i.e.*, the number of terms in the polynomial) per layer. More importantly, we compare against the state-of-the-art analytical method proposed by Qi et al. (2016), namely Paleo. To enable a comparison here and for the remainder of section, we executed the Paleo code[2] on the considered CNNs. We can easily observe that our predictions based on the layer-level models clearly outperform the best published model to date, yielding an *improvement in accuracy* up to 68.5%.

Table 2: Comparison of runtime models for common CNN layers – Our proposed runtime model consistently outperforms the state-of-the-art runtime model in both root-mean-square-error (RMSE) and the Root-Mean-Square-Percentage-Error (RMSPE).

Layer type	***NeuralPower***			Paleo Qi et al. (2016)	
	Model size	RMSPE	RMSE (ms)	RMSPE	RMSE (ms)
Convolutional	60	39.97%	1.019	58.29%	4.304
Fully-connected	17	41.92%	0.7474	73.76%	0.8265
Pooling	31	11.41%	0.0686	79.91%	1.763

Convolutional layer: The convolution layer is among the most time- and power-consuming components of a CNN. To model this layer, we use a polynomial model of degree three. We select a features vector consisting of the batch size, the input tensor size, the kernel size, the stride size, the padding size, and the output tensor size. In terms of the special terms defined in Equation 1, we use terms that represent the total computation operations and memory accesses count. **Fully-connected layer**: We employ a regression model with degree of two, and as features of the model we include the batch size, the input tensor size, and the output tensor size. It is worth noting that in terms of software implementation, there are two common ways to implement the fully-connected layer, either based on default matrix multiplication, or based on a convolutional-like implementation (*i.e.*, by keeping the kernel size exactly same as the input image patch size). Upon profiling, we notice that both cases have a tensor-reshaping stage when accepting intermediate results from a

2. Paleo is publicly available online – github repository: https://github.com/TalwalkarLab/paleo

previous convolutional layer, so we treat them interchangeably under a single formulation. **Pooling layer**: The pooling layer usually follows a convolution layer to reduce the complexity of the model. As basic model features we select the input tensor size, the stride size, the kernel size, and the output tensor size. Using Lasso and cross-validation we find that a polynomial of degree three provides the best accuracy.

4.1.3. Power Models

As mentioned in Section 3.1, we use the logarithmic terms of the original features (*e.g.*, batch size, kernel size, *etc.*) as additional features for the polynomial model since this significantly improves the model accuracy. This modeling choice is well suited for the nature of power consumption which does not scale linearly; more precisely, the rate of the increase in power goes down as the model complexity increases, especially when the power values get closer to the power budget limit. For instance, in our setup, the Titan X GPU platform has a maximum power of 250W. We find that a polynomial order of two achieves the best cross validation error for all three layer types under consideration.

To the best of our knowledge, there is no prior work on power prediction at the layer level to compare against. We therefore compare our methods directly with the actual power values collected from TensorFlow, as shown in Table 3. Once again, we observe that our proposed model formulation achieves error always less that 9% for all three layers. The slight increase in the model size compared to the runtime model is to be expected, given the inclusion of the logarithmic feature terms, alongside special terms that include memory accesses and operations count. We can observe, though, that the model is able to capture the trends of power consumption trained across layer sizes and types.

Table 3: Power model for common CNN layers

Layer type	***NeuralPower***		
	Model size	RMSPE	RMSE (W)
Convolutional	75	7.35%	10.9172
Fully-connected	15	9.00%	10.5868
Pooling	30	6.16%	6.8618

4.2. Network-level Modeling Evaluation

With the results from layer-level models, we can model the runtime, power, and energy for the whole network based on the network-level model (Section 3.2) in *NeuralPower*. To enable a comprehensive evaluation, we assess *NeuralPower* on several state-of-the-art CNNs, and we compare against the actual runtime, power, and energy values of each network. For this purpose, and as discussed in Section 3.3, we leave out a set of networks to be used only for testing, namely the VGG-16, NIN, CIFAR10-6conv, AlexNet, and Overfeat networks.

4.2.1. Runtime evaluation

Enabling network runtime profiling: Prior to assessing the predictions on the networks as a whole, we show the effectiveness of *NeuralPower* as a useful aid for CNN architecture benchmarking and per-layer profiling. Enabling such breakdown analysis is significant for machine learning practitioners, since it allows to identify the bottlenecks across components of a CNN.

For runtime, we use state-of-the-art analytical model Paleo as the baseline. In Figure 4, we compare runtime prediction models from *NeuralPower* and the baseline against actual runtime values of each layer in the NIN and VGG-16 networks. From Figure 4, we can clearly see that our model outperforms the Paleo model for most layers in accuracy. For the NIN, our model

clearly captures that *conv4* is the dominant (most time-consuming) layer across the whole network. However, Paleo erroneously identifies *conv2* as the dominant layer. For the VGG-16 network, we can clearly see that Paleo predicts the runtime of the first fully-connected layer *fc6* as 3.30 ms, with a percentage prediction error as high as -96.16%. In contrast, our prediction exhibits an error as low as -2.53%. Since layer *fc6* is the dominant layer throughout the network, it is critical to make a correct prediction on this layer.

From the above, we can conclude that our proposed methodology generally has a better accuracy in predicting the runtime for each layer in a complete CNN, especially for the layers with larger runtime values. Therefore, our accurate runtime predictions, when employed for profiling each layer at the network level, can help the machine learners and practitioners quickly identify the real bottlenecks with respect to runtime for a given CNN.

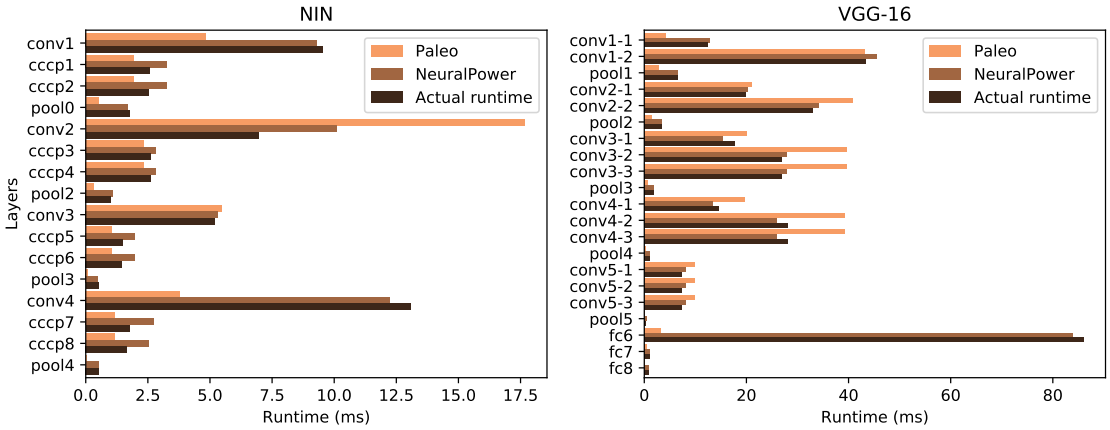

Figure 4: Comparison of runtime prediction for each layer in NIN and VGG-16: Our models provide accurate runtime breakdown of both network, while Paleo cannot. Our model captures the execution-bottleneck layers (*i.e.*, *conv4* in NIN, and *fc6* in VGG-16) while Paleo mispredicts both.

Network-level runtime evaluation: Having demonstrated the effectiveness of our methodology at the layer level, we proceed to assess the accuracy of the network-level runtime prediction $\hat{T}_{total}$ (Equation 3). It is worth observing that in Equation 3 there are two sources of potential error. First, error could result from mispredicting the runtime values $\hat{T}_n$ per layer n. However, even if these predictions are correct, another source of error could come from the formulation in Equation 3, where we assume that the sum of the runtime values of all the layers in a CNN provides a good estimate of the total runtime. Hence, to achieve a comprehensive evaluation of our modeling choices in terms of both the regression formulation and the summation in Equation 3, we need to address both these concerns.

To this end, we compare our runtime prediction $\hat{T}_{total}$ against two metrics. *First*, we compare against the actual overall runtime value of a network, notated as T_{total}. *Second*, we consider another metric defined as the sum of the actual runtime values T_n (and not the predictions) of each layer n:

$$\mathbb{T}_{total} = \sum_{n=1}^{N} T_n \tag{6}$$

Intuitively, a prediction value $\hat{T}_{total}$ close to both the $\mathbb{T}_{total}$ value and the actual runtime T_{total} would not only show that our model has good network-level prediction, but that also that our underlying modeling assumptions hold.

We summarize the results across five different networks in Table 4. More specifically, we show the networks' actual total runtime values (T_{total}), the runtime $\mathbb{T}_{total}$ values, our predictions $\hat{T}_{total}$, and the predictions from Paleo (the baseline). Based on the Table, we can draw two key observations. First, we can clearly see that our model always outperforms Paleo, with runtime predictions always within 24% from the actual runtime values. Unlike our model, prior art could underestimate the overall runtime up to 42%. Second, as expected, we see that summing the true runtime values per layer does indeed approximate the total runtime, hence confirming our assumption in Equation 3.

Table 4: Performance model comparison for the whole network. We can easily observe that our model always provides more accurate predictions of the total CNN runtime compared to the best published model to date (Paleo). We assess the effectiveness of our model in five different state-of-the-art CNN architectures.

CNN name	Qi et al. (2016) Paleo (ms)	***NeuralPower*** $\hat{T}_{total}$ (ms)	Sum of per-layer actual runtime $\mathbb{T}_{total}$ (ms)	Actual runtime T_{total} (ms)
VGG-16	345.83	373.82	375.20	368.42
AlexNet	33.16	43.41	42.19	39.02
NIN	45.68	62.62	55.83	50.66
Overfeat	114.71	195.21	200.75	197.99
CIFAR10-6conv	28.75	51.13	53.24	50.09

4.2.2. Power evaluation

Enabling power network profiling: We present a similar evaluation methodology to assess our model for network-level power predictions. We first use our methodology to enable a per-layer benchmarking of the power consumption. Figure 5 shows the comparison of our power predictions and the actual power values for each layer in the NIN and the VGG-16 networks. We can see that convolutional layers dominate in terms of power consumption, while pooling layers and fully connected layers contribute relatively less. We can also observe that the convolutional layer exhibits the largest variance with respect to power, with power values ranging from 85.80W up to 246.34W.

Another key observation is related to the fully-connected layers of the VGG-16 network. From Figure 4, we know layer *fc6* takes the longest time to run. Nonetheless, we can see in Figure 5 that its power consumption is relatively small. Therefore, the energy consumption related of layer *fc6* will have a smaller contribution to the total energy consumption of the network relatively to its runtime. It is therefore evident that using only the runtime as a proxy proportional to the energy consumption of CNNs could mislead the machine learners to erroneous assumptions. This observation highlights that power plays also a key role towards representative benchmarking of CNNs, hence illustrating further the significance of accurate power predictions enabled from our approach.

Table 5: Evaluating our power predictions for state-of-the-art CNN architectures.

CNN name	***NeuralPower*** $\hat{P}_{total}$ (W)	Sum of per-layer actual power $\mathbb{P}_{total}$ (W)	Actual power P_{avg} (W)
VGG-16	206.88	195.76	204.80
AlexNet	174.25	169.08	194.62
NIN	179.98	187.99	226.34
Overfeat	172.20	168.40	172.30
CIFAR10-6conv	165.33	167.86	188.34

Network-level power evaluation: As discussed in the runtime evaluation as well, we assess both our predictive model accuracy and the underlying assumptions in our formulation. In terms of average power consumption, we need to confirm that the formulation selected in Equation 4 is

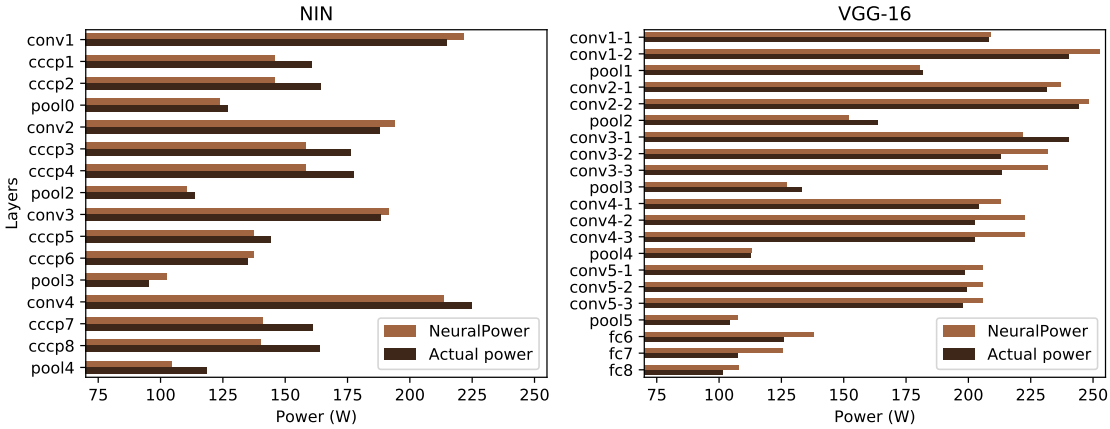

Figure 5: Comparison of power prediction for each layer in NIN and VGG-16.

indeed representative. To this end, besides the comparison against the actual average power of the network P_{avg}, we compare against the average value $\mathbb{P}_{avg}$, which can be computed by replacing our predictions $\hat{P}_n$ and $\hat{T}_n$ with with the actual per-layer runtime and power values:

$$\mathbb{P}_{avg} = \frac{\sum_{n=1}^{N} P_n \cdot T_n}{\sum_{n=1}^{N} T_n} \tag{7}$$

We evaluate our power value predictions for the same five state-of-the-art CNNs in Table 5. Compared to the actual power value, our prediction have an RMSPE of 11.66%. We observe that in two cases, AlexNet and NIN, our prediction has a larger error, *i.e.*, of 10.47% and 20.48% respectively. This is to be expected, since our formulation for $\mathbb{P}_{avg}$ depends on runtime prediction as well, and as observed previously in Table 4, we underestimate the runtime in both cases.

4.2.3. Energy evaluation

Finally, we use Equation 5 to predict the total energy based on our model. To evaluate our modeling assumptions as well, we compute the energy value $\mathbb{E}_{total}$ based on the actual per-layer runtime and power values, defined as:

$$\mathbb{E}_{total} = \sum_{n=1}^{N} P_n \cdot T_n \tag{8}$$

We present the results for the same five CNNs in Table 6. We observe that our approach enables good prediction, with an average RMSPE of 2.79%.

Table 6: Evaluating our energy predictions for state-of-the-art CNN architectures.

CNN name	***NeuralPower*** $\hat{E}_{total}$ (J)	Sum of per-layer actual energy $\mathbb{E}_{total}$ (J)	Actual energy E_{total} (J)
VGG-16	77.312	73.446	75.452
AlexNet	7.565	7.134	7.594
NIN	11.269	10.495	11.465
Overfeat	33.616	33.807	34.113
CIFAR10-6conv	8.938	8.453	9.433

4.3. Energy-Precision Ratio

In this subsection, we propose a new metric, namely *Energy-Precision Ratio* to be used as a guideline for machine learners towards accurate, yet energy efficient CNN architectures. We define the metric as:

$$M = Error^{\alpha} \cdot EPI \tag{9}$$

where $Error$ is the data classification error of the CNN under consideration, and EPI is the energy consumption per data item classified. Different values of the parameter α dictate how much importance is placed on the accuracy of the model, since a larger α places more weight on the CNN classification error. To illustrate how α affects the results, in Table 7 we compute the M score values for VGG-16, AlexNet, and NIN, all trained on ImageNet datasets (as $Error$ value we use their Top-5 error). We also use our predictive model for energy, and we compute the energy consumption per image classification.

Intuitively, a lower M value indicates a better trade-off between energy-efficiency and accuracy of a CNN architecture. For instance, we can see that while VGG-16 has the lowest error, this comes at the price of increased energy consumption compared to both NIN and AlexNet. Hence, for $\alpha = 1$ both AlexNet and NIN have a smaller M value. In this case, a machine learner of an embedded Internet-of-Things (IoT) system could use the *Energy-Precision Ratio* to select the most energy efficient architecture. On the contrary, with $\alpha = 4$, *i.e.*, when accuracy is being heavily favored over energy efficiency, the M value of VGG-16 is smaller than M value of AlexNet.

Table 7: M metric for different CNN architectures and Energy-per-Image (EPI) values. Network choices could be different for different α values: AlexNet for $\alpha = 1, 2, 3$, VGG-16 for $\alpha = 4$.

CNN name	Top-5 Error	EPI (mJ)	M			
			$\alpha = 1$	$\alpha = 2$	$\alpha = 3$	$\alpha = 4$
VGG-16	7.30%	1178.9	86.062	6.283	0.459	**0.033**
AlexNet	17.00%	59.3	**10.086**	**1.715**	**0.291**	0.050
NIN	20.20%	89.6	18.093	3.655	0.738	0.149

With the recent surge of IoT and mobile learning applications, machine learners need to take the energy efficiency of different candidate CNN architectures into consideration, in order to identify the CNN which is more suitable to be deployed on a energy-constrained platform. For instance, consider the case of a machine learner that has to choose among numerous CNNs from a CNN model "zoo", with the best error of each CNN readily available. *Which network provides the best trade-off between accuracy for energy spent per image classified?* Traditionally, the main criterion for choosing among CNN architectures has been the data item classification accuracy, given its intuitive nature as a metric. However, there has not been so far an easily interpretable metric to trade-off energy efficiency versus accuracy.

Towards this direction, we can use our proposed model to quickly predict the total energy consumptions of all these different architectures, and we can then compute the M score to select the one that properly trades off between accuracy for energy spent per image classified. We postulate that the *Energy-Precision Ratio*, alongside our predictive model, could therefore be used as a useful aid for machine learners *towards energy-efficient CNNs*.

4.4. Models on other platforms

Our key goal is to provide a modeling framework that could be flexibly used for different hardware and software platforms. To comprehensively demonstrate this property of our work, we extend our

evaluation to a different GPU platform, namely Nvidia GTX 1070, and another Deep Learning software tool, namely Caffe by Jia et al. (2014).

4.4.1. Extending to other hardware platforms: Tensorflow on Nvidia GTX 1070

We first apply our framework to another GPU platform, and more specifically to the Nvidia GTX 1070 with 6GB memory. We repeat the runtime and power data collection by executing Tensorflow, and we train power and runtime models on this platform. The layer-wise evaluation results are shown in Table 8. For these results, we used the same polynomial orders as reported previously for the TensorFlow on Titan X experiments. Moreover, we evaluate the overall network prediction for runtime, power, and energy values and we present the predicted values and the prediction error (denoted as Error) in Table 9. Based on these results, we can see that our methodology achieves consistent performance across different GPU platforms, thus enabling a scalable/portable framework from machine learning practitioners to use across different systems.

Table 8: Runtime and power model for all layers using TensorFlow on GTX 1070.

Layer type	Runtime			Power		
	Model size	RMSPE	RMSE (ms)	Model size	RMSPE	RMSE (W)
Convolutional	10	57.38 %	3.5261	52	10.23%	9.4097
Fully-connected	18	44.50%	0.4929	23	7.08%	5.5417
Pooling	31	11.23%	0.0581	40	7.37%	5.1666

Table 9: Evaluation of *NeuralPower* on CNN architectures using TensorFlow on GTX 1070.

CNN name	Runtime		Power		Energy	
	Value (ms)	Error	Value (W)	Error	Value (J)	Error
AlexNet	44.82	17.40%	121.21	-2.92%	5.44	13.98%
NIN	61.08	7.24%	120.92	-4.13%	7.39	2.81%

4.4.2. Extending to other machine learning software environments: Caffe

Finally, we demonstrate the effectiveness of our approach across different Deep Learning software packages and we replicate our exploration on Caffe. To collect the power and runtime data for our predictive models, we use Caffe's supported mode of operation for benchmarking, namely `time`. While this functionality benchmarks the model execution layer-by-layer, the default Caffe version however reports only timing. To this end, we extend Caffe's `C++` codebase and we incorporate calls to Nvidia's `NVML C++ API` to report power values.

We present the per-layer accuracy for runtime and power predictions in Table 10. Furthermore, we evaluate our model on the AlexNet and NIN networks in Table 11. Please note that the execution of the entire network corresponds to a different routine under the Caffe framework, so a direct comparison is not possible. We instead compare against the Equations 6-8 as in the previous subsection. Same as for the TensorFlow case before (Table 8), we observe that in this case as well the runtime predictions exhibit a larger error in this platform when executing on the GTX1070 system. This is to be expected, since the GTX 1070 drivers do not allow the user to clock the GPU to a particular frequency state, hence the system dynamically selects its execution state. Indeed, in our collected datapoints we observed that Caffe's (and TensorFlow's previously) higher variance in the runtime values. To properly capture this behavior, we experimented with regressors other that second and third degree polynomials. In these results for Caffe in particular, we select linear models since they provided a better trade off between training error and overfitting.

Table 10: Runtime and power model for all layers using Caffe on GTX 1070.

Layer type	Runtime			Power		
	Model size	RMSPE	RMSE (ms)	Model size	RMSPE	RMSE (W)
Convolutional	32	45.58 %	2.2301	32	6.19%	11.9082
Fully-connected	18	48.41 %	0.6626	18	8.63%	8.0291
Pooling	30	37.38 %	0.1711	26	6.72 %	11.9124

Table 11: Evaluation of our model on CNN architectures using Caffe on GTX 1070.

CNN name	Runtime		Power		Energy	
	Value (ms)	Error	Value (W)	Error	Value (J)	Error
AlexNet	51.18	-31.97%	107.63	-5.07 %	5.51	35.42%
NIN	76.32	0.36 %	109.78	-8.89%	8.38	8.56%

4.5. Discussion

It is important to note that overhead introduced by *NeuralPower* is not significant. More specifically, *NeuralPower* needs to collect datasets to train the models, however, the overhead for training is very small, e.g., around 30 minutes for GPU Titan X. This includes data collection (under 10 minutes) and model training (less than 20 minutes). The process is done once for a new platform. This overhead can be easily offset if the CNN architecture search space is large. Even if machine learners only evaluate a few CNN architectures, *NeuralPower* can still provide the detailed breakdown with respect to runtime, power, and energy to identify bottlenecks and possible improvement directions.

5. Conclusion

With the increased popularity of CNN models, the runtime, power, and energy consumption have emerged as key design issues when determining the network architecture or configurations. In this work, we propose *NeuralPower*, the first holistic framework to provide an accurate estimate of power, runtime, and energy consumption. The runtime model of *NeuralPower* outperforms the current state-of-the-art predictive model in terms of accuracy. Furthermore, *NeuralPower* can be used to provide an accurate breakdown of a CNN network, helping machine learners identify the bottlenecks of their designed CNN models. Finally, we assess the accuracy of predictions at the network level, by predicting the runtime, power, and energy of state-of-the-art CNN configurations. *NeuralPower* achieves an average accuracy of 88.24% in runtime, 88.34% in power, and 97.21% in energy. As future work, we aim to extend our current framework to model the runtime, power, and energy of the networks with more sophisticated parallel structures, such as the ResNet network by He et al. (2016).

Acknowledgments: This research was supported in part by NSF CCF Grant No. 1514206 and NSF CNS Grant No. 1564022.

References

Matthieu Courbariaux, Yoshua Bengio, and Jean-Pierre David. Binaryconnect: Training deep neural networks with binary weights during propagations. In *Advances in Neural Information Processing Systems*, pages 3123–3131, 2015.

Matthieu Courbariaux, Itay Hubara, Daniel Soudry, Ran El-Yaniv, and Yoshua Bengio. Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1. *arXiv preprint arXiv:1602.02830*, 2016.

Song Han, Jeff Pool, John Tran, and William Dally. Learning both weights and connections for efficient neural network. In *Advances in Neural Information Processing Systems*, pages 1135–1143, 2015.

Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

Yangqing Jia, Evan Shelhamer, Jeff Donahue, Sergey Karayev, Jonathan Long, Ross Girshick, Sergio Guadarrama, and Trevor Darrell. Caffe: Convolutional architecture for fast feature embedding. *arXiv preprint arXiv:1408.5093*, 2014.

Alex Krizhevsky. One weird trick for parallelizing convolutional neural networks. *arXiv preprint arXiv:1404.5997*, 2014.

Yann LeCun, Léon Bottou, Yoshua Bengio, and Patrick Haffner. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11):2278–2324, 1998.

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

Hang Qi, Evan R Sparks, and Ameet Talwalkar. Paleo: A performance model for deep neural networks. 2016.

Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun. Faster r-cnn: Towards real-time object detection with region proposal networks. In *Advances in neural information processing systems*, pages 91–99, 2015.

Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Delight: Adding energy dimension to deep neural networks. In *Proceedings of the 2016 International Symposium on Low Power Electronics and Design*, pages 112–117. ACM, 2016a.

Bita Darvish Rouhani, Azalia Mirhoseini, and Farinaz Koushanfar. Going deeper than deep learning for massive data analytics under physical constraints. In *Proceedings of the Eleventh IEEE/ACM/IFIP International Conference on Hardware/Software Codesign and System Synthesis*, page 17. ACM, 2016b.

Pierre Sermanet, David Eigen, Xiang Zhang, Michaël Mathieu, Rob Fergus, and Yann LeCun. Overfeat: Integrated recognition, localization and detection using convolutional networks. *arXiv preprint arXiv:1312.6229*, 2013.

Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

SC Smithsons, G Yang, WJ Gross, and WJ Gross. Neural networks designing neural networks: Multi-objective hyper-parameter optimization. *arXiv preprint arXiv:1611.02120*, 2016.

Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 1–9, 2015.

Chen Zhang, Peng Li, Guangyu Sun, Yijin Guan, Bingjun Xiao, and Jason Cong. Optimizing fpga-based accelerator design for deep convolutional neural networks. In *Proceedings of the 2015 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, FPGA '15, pages 161–170, 2015a. ISBN 978-1-4503-3315-3.

Qian Zhang, Ting Wang, Ye Tian, Feng Yuan, and Qiang Xu. Approxann: an approximate computing framework for artificial neural network. In *2016 Design, Automation and Test in Europe Conference and Exhibition*, pages 701–706. IEEE, 2015b.

Barret Zoph and Quoc V Le. Neural architecture search with reinforcement learning. *arXiv preprint arXiv:1611.01578*, 2016.