
FermiNets: Learning generative machines to generate efficient neural networks via generative synthesis

Alexander Wong^{1,2}, Mohammad Javad Shafiee^{1,2}, Brendan Chwyl², and Francis Li²

¹Waterloo Artificial Intelligence Institute, University of Waterloo, Waterloo, ON, Canada

²DarwinAI Corp., Waterloo, ON, Canada

{a28wong,mjshafiee}@uwaterloo.ca, {francis,brendan}@darwin.ca

Abstract

The tremendous potential exhibited by deep learning is often offset by architectural and computational complexity, making widespread deployment a challenge for edge scenarios such as mobile and other consumer devices. To tackle this challenge, we explore the following idea: **Can we learn generative machines to automatically generate deep neural networks with efficient network architectures?** In this study, we introduce the idea of **generative synthesis**, which is premised on the intricate interplay between a generator-inquisitor pair that work in tandem to garner insights and learn to generate highly efficient deep neural networks that best satisfies operational requirements. What is most interesting is that, once a generator has been learned through generative synthesis, it can be used to generate not just one but a large variety of different, unique highly efficient deep neural networks that satisfy operational requirements. Experimental results for image classification, semantic segmentation, and object detection tasks illustrate the efficacy of generative synthesis in producing generators that automatically generate highly efficient deep neural networks (which we nickname **FermiNets**) with higher model efficiency and lower computational costs (reaching $>10\times$ more efficient and fewer multiply-accumulate operations than several tested state-of-the-art networks), as well as higher energy efficiency (reaching $>4\times$ improvements in image inferences per joule consumed on a Nvidia Tegra X2 mobile processor). As such, generative synthesis can be a powerful, generalized approach for accelerating and improving the building of deep neural networks for on-device edge scenarios.

1 Introduction

Deep learning [12] has garnered tremendous attention, driven by recent breakthroughs in a wide range of applications such as image categorization [11] and speech recognition [5]. Despite these successes, the increasing complexities of deep neural networks limit their widespread adoption in edge scenarios such as mobile and other consumer devices where computational, memory, bandwidth, and energy resources are scarce. There has in recent years been a noticeable increase in the exploration of strategies to improve the efficiency of deep neural networks, particularly on edge and mobile devices. One common approach explored is precision reduction [10, 14, 3], in which the data representation of a network is reduced from typical 32-bit floating point precision to fixed-point or integer precision [10], 2-bit precision [14], or even 1-bit precision [3]. Another strategy explored is model compression [4, 7, 15], which involves traditional data compression methods such as weight thresholding, hashing, and Huffman coding, as well as teacher-student strategies involving a larger teacher network training a smaller student network. Finally, another technique that has been investigated targets the fundamental design of deep neural networks and involves leveraging architectural design principles to achieve more efficient deep neural network macroarchitectures [8, 9, 17, 21].

In this study, we explore a very different idea: **Can we learn generative machines to automatically generate deep neural networks with efficient network architectures?** We introduce the idea of **generative synthesis**, which is premised on the intricate interplay between a generator-inquisitor pair that work in tandem to garner insights and learn to generate highly efficient deep neural networks that best satisfies operational requirements, such as those needed for on-device edge scenarios.

2 Generative Synthesis

For the sake of brevity, a brief mathematical description of the methodology behind generative synthesis is provided as follows. Let a deep neural network be defined as a graph $N = \{V, E\}$, comprising of a set V of vertices $v \in V$ and a set E of edges $e \in E$ that form the network. Now, let a generator $\mathcal{G}(s; \theta_{\mathcal{G}})$ be defined as a function parameterized by $\theta_{\mathcal{G}}$ that, given a seed $s \in S$, generates a deep neural network $N_s = \{V_s, E_s\}$ (i.e., $N_s = \mathcal{G}(s)$), where S is the set of possible seeds. Given operational requirements (characterized by indicator function $1_r(\cdot)$), the goal of generative synthesis is to learn an expression of \mathcal{G} that can generate deep neural networks $\{N_s | s \in S\}$ that maximize a universal performance function \mathcal{U} (e.g., [20]) while satisfying requirements defined by $1_r(\cdot)$. One can thus formulate the problem of learning \mathcal{G} as the following constrained optimization problem,

$$\mathcal{G} = \max_{\mathcal{G}} \mathcal{U}(\mathcal{G}(s)) \text{ subject to } 1_r(\mathcal{G}(s)) = 1, \forall s \in S. \quad (1)$$

Solving the problem posed in Eq. 1 is highly non-trivial given the enormity of the feasible region, and thus an efficient approach for solving this optimization problem is highly desired.

In *generative synthesis*, we find an approximate solution to the problem posed in Eq. 1 by leveraging the interplay between a generator-inquisitor pair $\{\mathcal{G}, \mathcal{I}\}$, with \mathcal{G} denoting a generator and \mathcal{I} denoting an inquisitor that work in tandem to obtain improved insights about deep neural networks as well as learn to generate highly efficient networks in a cyclical manner. More specifically, let $\mathcal{I}(\mathcal{G}; \theta_{\mathcal{I}})$ be defined as a function parameterized by $\theta_{\mathcal{I}}$ that, given a generator \mathcal{G} , produces a set of parameter changes $\Delta\theta_{\mathcal{G}}$ (i.e., $\Delta\theta_{\mathcal{G}} = \mathcal{I}(\mathcal{G})$). At initialization, both $\theta_{\mathcal{G}}$ and $\theta_{\mathcal{I}}$ are initialized based on prototype φ , \mathcal{U} , and $1_r(\cdot)$, resulting in \mathcal{G}_0 and \mathcal{I}_0 . After initialization, at each cycle k , the generator at cycle k (i.e., \mathcal{G}_k) generates a new deep neural network N_{s^k} based on a generated seed s_k (i.e., $N_{s^k} = \mathcal{G}_k(s_k)$).

After the generation of N_{s^k} using \mathcal{G}_k , $\{\mathcal{V}_{s^k}, \mathcal{E}_{s^k}\}$ in N_{s^k} are probed with a set X of targeted stimulus signals $x \in X$ and the corresponding set $Y_{\mathcal{G}_k(s_k)}$ of reactionary response signals $y \in Y_{\mathcal{G}_k(s_k)}$ are observed, where $\mathcal{V}_{s^k} \subseteq V_{s^k}$ and $\mathcal{E}_{s^k} \subseteq E_{s^k}$. After the observation process, $\theta_{\mathcal{I}}$ is updated based on $Y_{\mathcal{G}_k(s_k)}$, $\mathcal{U}(\mathcal{G}_k(s_k))$, and $1_r(\mathcal{G}_k(s_k))$ to obtain \mathcal{I}_{k+1} . In particular, by probing $\{\mathcal{V}_{s^k}, \mathcal{E}_{s^k}\}$ in N_{s^k} with X and observing $Y_{\mathcal{G}_k(s_k)}$, the inquisitor \mathcal{I} is able to learn at a foundational level about the architectural efficiencies of N_{s^k} via information-theoretic insights that are derived from $Y_{\mathcal{G}_k(s_k)}$.

After the inquisitor update process, $\theta_{\mathcal{G}}$ is updated according to $\Delta\theta_{\mathcal{G}_k(s_k)}$ produced by \mathcal{I}_{k+1} to obtain \mathcal{G}_{k+1} . The aforementioned process of generating, probing, observation, and updating is repeated over cycles, resulting in a sequence of improving approximate solutions of \mathcal{G} to the problem in Eq. 1. What is most interesting about generative synthesis is that, once a generator \mathcal{G} has been learned, it can be used to generate not just one but a large variety of different, unique highly efficient deep neural networks N , using different seeds $s \in S$, that satisfy operational requirements defined by $1_r(\cdot)$.

3 Experimental Results and Discussion

To evaluate the efficacy of generative synthesis (which we will refer to as **GenSynth** for short from here on) in producing generators that automatically generate highly efficient deep neural networks (which we nickname **FermiNets** because of their very small sizes), three experiments were performed:

- **Image classification.** φ : ResNet [6], $1_r(\cdot)$: accuracy on CIFAR-10 $\geq 89\%$.
- **Semantic segmentation.** φ : RefineNet [13], $1_r(\cdot)$: accuracy on CamVid [1] $\geq 90\%$.
- **Object detection.** φ : DetectNet [19], $1_r(\cdot)$: mAP on Parse27K [18] $\geq 61\%$.

The performance of the generated FermiNets was evaluated using these metrics: i) **information density** [2] as a metric for assessing model efficiency, ii) **multiply-accumulate (MAC) operations** as a metric for computational cost, and iii) **NetScore** [20] as a metric for assessing overall network performance (balance between accuracy, architectural complexity, and computational complexity).

Image classification: The top-1 test accuracy of the generated FermiNets along with MobileNet [8], ShuffleNet [21], and NASNet-L2C(S) [16] is shown in Fig. 1 (top left), with FermiNet-A, FermiNet-B, and FermiNet-C providing the highest top-1 accuracies amongst the tested state-of-the-art efficient networks ($\sim 1.4\%$, $\sim 0.4\%$, and $\sim 0.01\%$ higher than NASNet-L2C(S), respectively). As shown

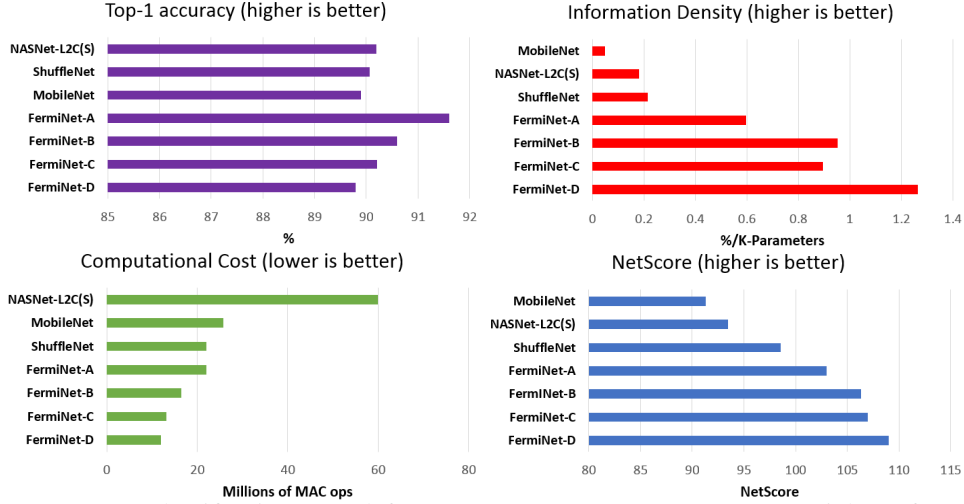


Figure 1: Image classification: (top left) Top-1 accuracy on CIFAR-10, (top right) Information density [2], (bottom left) MAC operations, and (bottom right) NetScore [20]

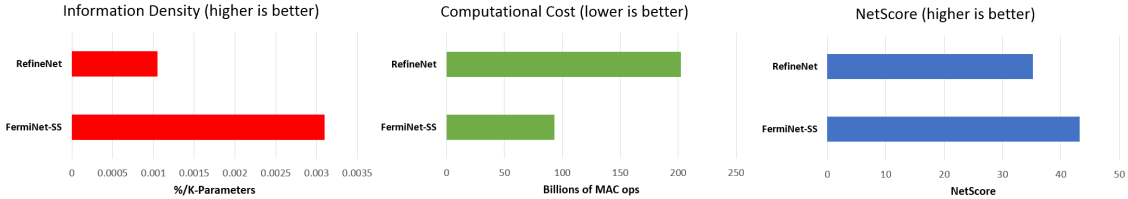


Figure 2: Semantic segmentation: (a) Information density, (b) MAC operations, and (c) NetScore.

in Fig. 1 (top right), the information densities of the generated FermiNets had noticeably higher information densities than MobileNet, ShuffleNet, as well as NASNet-L2C(S)¹ which was produced via a combination of state-of-the-art model compression approaches [10, 7, 15]. In particular, FermiNet-A, FermiNet-B, FermiNet-C, and FermiNet-D have information densities that exceeds that of MobileNet by $>12.5\times$, $\sim 20\times$, $\sim 19\times$, and $>26.4\times$, respectively. As shown in Fig. 1 (bottom left), the FermiNets require noticeably fewer number of MAC operations than the other tested networks, with that of FermiNet-A, FermiNet-B, FermiNet-C, and FermiNet-D less than that of NASNet-L2C(S) by $>2.7\times$, $>3.6\times$, $>4.5\times$, and $\sim 5\times$, respectively. To evaluate overall network performance, the NetScore of the tested networks is shown in Fig. 1 (bottom right), with the FermiNets having noticeably higher NetScores when compared to MobileNet, ShuffleNet, and NASNet-L2C(S). In particular, the NetScore of FermiNet-A, FermiNet-B, FermiNet-C, and FermiNet-D exceed that of MobileNet by >11.8 points, >15 points, >15.6 , and >17.5 points, respectively. These significant improvements in NetScore achieved by the generated FermiNets illustrate the power of GenSynth in striking a strong balance between accuracy, architectural complexity, and computational cost, which is important for on-device edge scenarios.

Semantic segmentation: The pixel-wise accuracy of RefineNet [13] and the generated FermiNet-SS are 90.3% and 90.2%, respectively. As shown in Fig. 2(a), the information density of FermiNet-SS exceeds that of RefineNet by $\sim 3\times$, thus illustrate its modelling efficiency. As shown in Fig. 2(b), the number of MAC operations used by FermiNet-SS is less than that of RefineNet by $\sim 2.2\times$. Finally, as shown in Fig. 2(c), the NetScore of FermiNet-SS exceeds that of RefineNet by ~ 8 points.

Object detection: The mean average precision (mAP) of DetectNet [19] and the generated FermiNet-OD are 61.8% and 61.0%, respectively. As shown in Fig. 3(a), the information density of FermiNet-OD exceeds that of DetectNet by $>10\times$. The number of MAC operations used by FermiNet-OD along with that of DetectNet is shown in Fig. 3(b). It can be observed that the number of MAC operations used by FermiNet-OD less than that of DetectNet by $>11\times$. As shown in Fig. 3(c), the NetScore of FermiNet-OD exceeds that of DetectNet by >21 points. In an additional experiment, we study the energy efficiency of FermiNet-OD when operating in edge device scenarios. To quantitatively assess the energy efficiency during inference, the metric we leverage in this study is the number of image

¹Note that, since the number of parameters for NASNet-L2C(S) was not reported in [16], it was approximated based on the reported model size to enable the computation of information density and NetScore.

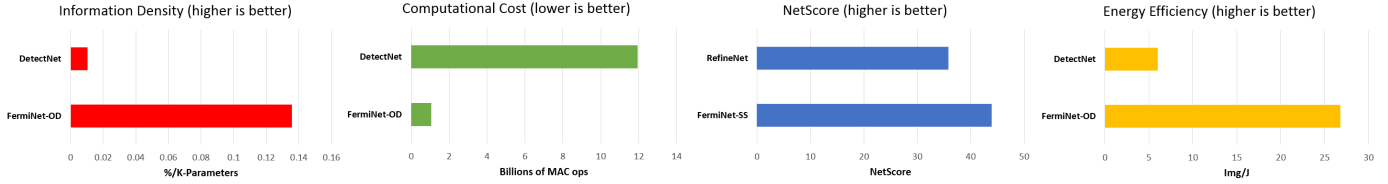


Figure 3: Object detection: (a) Information density, (b) MAC operations, (c) NetScore, and (d) energy

inferences per joule consumed (img/J) by a Nvidia Tegra X2 mobile processor. As shown in Fig. 3(d), FermiNet-OD is significantly more energy efficient than DetectNet, enabling the Tegra X2 mobile processor to process $>4\times$ more images per joule than can be achieved with DetectNet.

As shown by the empirical results in this study, it can be seen that GenSynth can be a powerful, generalized approach for building deep neural networks that satisfies operational requirements for on-device edge scenarios such as mobile and other consumer devices.

Acknowledgements

The authors thank Akif Kamal, Michael St. Jules, Stanislav Bochkarev, David Dolson, Xiao Yu Wang, and Desmond Lin at DarwinAI Corp. for their support and assistance in experimental preparations.

References

- [1] G. Brostow et al. Semantic object classes in video: A high-definition ground truth database. In *PRL*, 2008.
- [2] A. Canziani et al. An analysis of deep neural network models for practical applications. *arXiv:1605.07678*, 2017.
- [3] M. Courbariaux et al. BinaryConnect: Training deep neural networks with binary weights during propagations. *NIPS*, 2015.
- [4] S. Han et al. Deep Compression: Compressing deep neural networks with pruning, trained quantization and huffman coding. *arXiv:1510.00149*, 2015.
- [5] A. Hannun et al. Deep Speech: Scaling up end-to-end speech recognition. *arXiv:1412.5567*, 2014.
- [6] K. He et al. Deep residual learning for image recognition. *arXiv:1512.03385*, 2015.
- [7] G. Hinton et al. Distilling the knowledge in a neural network. *arXiv:1503.02531*, 2015.
- [8] A. Howard et al. MobileNets: Efficient convolutional neural networks for mobile vision applications. *arXiv:1704.04861*, 2017.
- [9] F. Iandola et al. SqueezeNet: Alexnet-level accuracy with 50x fewer parameters and $<0.5\text{mb}$ model size. *arXiv:1602.07360*, 2016.
- [10] B. Jacob et al. Quantization and training of neural networks for efficient integer-arithmetic-only inference. *arXiv:1712.05877*, 2017.
- [11] A. Krizhevsky et al. Imagenet classification with deep convolutional neural networks. *NIPS*, 2012.
- [12] Y. LeCun, Y. Bengio, and G. Hinton. Deep learning. *Nature*, 2015.
- [13] G. Lin et al. Refinenet: Multi-path refinement networks for high-resolution semantic segmentation. *arXiv:1611.06612*, 2016.
- [14] W. Meng et al. Two-bit networks for deep learning on resource-constrained embedded devices. *arXiv:1701.00485*, 2017.
- [15] S. Ravi. ProjectionNet: Learning efficient on-device deep networks using neural projections. *arXiv:1708.00630*, 2017.
- [16] S. Ravi et al. Custom on-device ML models with Learn2Compress. <https://ai.googleblog.com/2018/05/custom-on-device-ml-models.html>, 2018.
- [17] M. Shafiee et al. SquishedNets: Squishing SqueezeNet further for edge device scenarios via deep evolutionary synthesis. *NIPS*, 2017.
- [18] P. Sudowe et al. Person Attribute Recognition with a Jointly-trained Holistic CNN Model. In *ICCV ChaLearn Looking at People Workshop*, 2015.
- [19] A. Tao et al. DetectNet: Deep neural network for object detection in DIGITS. In *Nvidia*, 2016.
- [20] A. Wong. NetScore: Towards universal metrics for large-scale performance analysis of deep neural networks for practical usage. *arXiv:1806.05512*, 2018.
- [21] X. Zhang et al. ShuffleNet: An extremely efficient convolutional neural network for mobile devices. *arXiv:1707.01083*, 2017.