NEURAL NETWORK TRAINING WITH APPROXIMATE LOGARITHMIC COMPUTATIONS

Arnab Sanyal, Peter A. Beerel, and Keith M. Chugg

Ming Hsieh Department of Electrical and Computer Engineering
University of Southern California
Los Angeles, California 90089, USA
{arnabsan, pabeerel, chugg}@usc.edu

ABSTRACT

The high computational complexity associated with training deep neural networks limits online and real-time training on edge devices. This paper proposed an end-to-end training and inference scheme that eliminates multiplications by approximate operations in the log-domain which has the potential to significantly reduce implementation complexity. We implement the entire training procedure in the log-domain, with fixed-point data representations. This training procedure is inspired by hardware-friendly approximations of log-domain addition which are based on look-up tables and bit-shifts. We show that our 16-bit log-based training can achieve classification accuracy within approximately 1% of the equivalent floating-point baselines for a number of commonly used datasets.

Index Terms— Logarithmic Number System, Deep Neural Networks, Approximate Computation

1. INTRODUCTION

In recent years neural networks with hidden layers, or *deep neural networks* (DNNs), have found widespread application in a large number of pattern recognition problems, notably speech recognition and computer vision [1]. This resurgence in interest and application of neural networks has been driven by the availability of large data sets and increased computation resources. In particular, *graphic processor units* (GPUs) provide a large number of hardware *multiply-accumulate* (MAC) accelerators and are therefore widely used in the MAC-intensive process of training DNNs.

Despite these advances, custom hardware accelerators have the potential to further improve the speed and energy efficiency in DNN training and inference modes. In a custom implementation one can co-design the computational circuitry, the simple control circuitry, and the memory architecture to achieve near full utilization of the hardware. This provides a potential advantage over GPUs which are more general purpose and may not efficiently utilize all the associated MAC hardware units during DNN processing. Most researchers have focused on accelerating inference processing with an eye towards using trained DNNs on edge computing devices (cf., [2]) with a few more recent investigations [3, 4] considering accelerated training. Hardware acceleration of training has the potential to reduce energy consumption in data centers and to provide learning directly on edge devices.

There are a wide class of DNN complexity reduction methods based on sparsity, pruning, and quantization (cf., [5, 6, 7]). Complimentary to these approaches are methods of reducing the complexity of MAC units. Motivated by the fact that multiplier circuitry dominates the complexity of MAC units, several researchers have studied the use of *logarithmic number system* (LNS) [8, 9] wherein multiplications are replaced with additions. LNS methods have been proposed in communications [10], processor design [11], re-configurable architectures [12], and a number of signal processing applications [13]. The primary challenge of LNS-based MAC processing is the log-domain addition operation which has generally been addressed with functional approximation or *look-up tables* LUTs. Preliminary work [14, 15] proposed an 8 bit input, 16 bit output LNS-based MAC using LUTs and claimed a $3.2\times$ improvement in area-delay product compared to an equivalent linear-domain MAC. This was investigated in the context of back-propagation, but was prior to resurgence of neural networks and therefore was not investigated in the context of modern datasets, larger networks, and other modern deep learning methods and conventions. Previous work also studied encoding weights in LNS for inference [16] and proposed extensions to LNS MACs restricted to positive numbers [17]. A more recent paper [18] implemented log-encoding on posits [19] but relied on conversions to and from the linear domain to perform addition. Recent work has demonstrated LNS circuit implementation for inference in a recurrent neural network that was pre-trained off-line using traditional methods.[20].

In this paper we propose end-to-end log-based training and inference of DNNs using approximate LNS computations. We generalize the bit-shift approximation in [17] to handle signed arithmetic and show that these bit-shift approximations are special cases of a LUT. To evaluate this approach we train a number of networks using fixed-point data repre-

This work is supported in part by the National Science Foundation (CCF-1763747).

sentations in both the conventional linear domain and using our proposed approximate LNS computations based on both LUT and bit-shift approximations. Our results show that with 16-bit words and a 20-element LUT the degradation in training accuracy, relative to conventional floating point linear processing, is small (i.e., $\lesssim 1\%$ loss in accuracy). Thus, this work demonstrates the feasibility of multiply-free, approximate LNS training and inference; a detailed analysis of the potential complexity reduction in digital circuitry is beyond the scope of this work.

The remainder of this paper is organized as follows - A brief description of LNS is provided in Section 2. Several approximations for LNS operations are developed in Section 3. Section 4 contains a description of the end-to-end training scheme of a neural network in log-domain as well as analysis relating bit-widths for fixed-point processing in the linear and log domains. Experimental results are summarized in Section 5 and conclusions provided in Section 6.

2. LOGARITHMIC NUMBER SYSTEM

In a LNS, a real number v is represented by the logarithm of its absolute value and its sign. Thus,

$$v \longleftrightarrow \underline{V} = (V, s_v) \tag{1a}$$

$$V = \log_2(|v|) \tag{1b}$$

$$s_v = \operatorname{sign}(v) \tag{1c}$$

where $\operatorname{sign}(v) = 1$ if $v > 0$ and 0 otherwise. Note that the radix of the logarithm does not change the important properties of LNS, but using radix 2 leads to bit-shift approximations as described in Section 3. Multiplication in the linear-domain becomes addition in log-domain

$$u = xy \longleftrightarrow \underline{U} = \underline{X} \boxdot \underline{Y} \tag{2a}$$

$$U = X + Y \tag{2b}$$

$$s_u = \overline{(s_x \veebar s_y)} \tag{2c}$$

where $\veebar$ denotes the exclusive OR operation and $\overline{s}$ denotes the compliment of the binary variable s. We define addition in log-domain as follows

$$z = x + y \longleftrightarrow \underline{Z} = \underline{X} \boxplus \underline{Y} \tag{3a}$$

$$Z = \begin{cases} \max(X,Y) + \Delta_+(|X-Y|) & s_x = s_y \\ \max(X,Y) + \Delta_-(|X-Y|) & s_x \neq s_y \end{cases} \tag{3b}$$

$$s_z = \begin{cases} s_x & X > Y \\ s_y & X \leq Y \end{cases} \tag{3c}$$

where the Δ terms exact representation of addition in log-domain in the extended real numbers R^+. The functional expression of Δ terms are

$$\Delta_+(d) = \log_2\left(1 + 2^{-d}\right) \qquad d \geq 0 \tag{4a}$$

$$\Delta_-(d) = \log_2\left(1 - 2^{-d}\right) \qquad d \geq 0 \tag{4b}$$

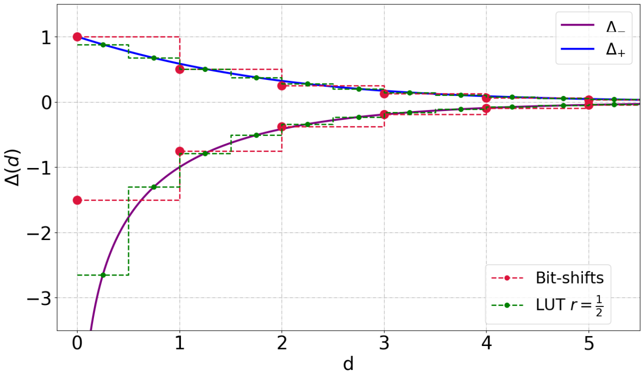


Fig. 1: Approximation to $\Delta_\pm(d)$ (table size is 20).

To reduce the computational complexity of calculating Δ, we describe two different approximations in Section 3 that induce approximate addition in the log-domain. One can extend the above concepts to define log-domain subtraction as

$$t = x - y \longleftrightarrow \underline{T} = \underline{X} \boxminus \underline{Y} = \underline{X} \boxplus (Y, \overline{s_y}) \tag{5}$$

Exponentiation translates to a multiplication in the log-domain. This operation is non-commutative as in general $a^b \neq b^a$. We define log-domain exponentiation when exponentiating on a positive radix $x > 0$,

$$w = x^y \longleftrightarrow \underline{W} = (yX, 1) \tag{6}$$

3. APPROXIMATE LOG-DOMAIN ADDITION

It is clear from (2) that LNS processing reduces the complexity of multiplication, but the Δ terms in (3) associated with log-domain addition are much more complex to implement than standard addition. Motivated by the fact that the training process is inherently noisy (e.g., gradient noise, finite precision effects, etc.) we propose low-complexity approximations of the Δ terms in (3). Specifically we seek the simplest such approximations that do not significantly degrade the overall accuracy of the trained networks. Look-up tables provide a natural approach to approximating the Δ terms. In this paper we consider simple LUTs for $\Delta_\pm(d)$ wherein the *dynamic range* of d supported is $[0, d_{\max}]$ and the resolution is r. Specifically, each unit interval within the dynamic range has $1/r$ points uniformly sampled from $\Delta_\pm(d)$. This concept is shown in Fig. 1. Note that the size of the LUT is $d_{\max}/r$.

A bit-shift approximation for $\Delta_+(d)$ was suggested in [17]. This can be generalized using

$$\log_e(1 \pm x) \approx \pm x \qquad 0 \leq x \ll 1 \tag{7}$$

which, together with (4) implies that

$$\Delta_\pm(d) \approx \pm 2^{-d}\left(1 + 2^{-1} - 2^{-4} + \cdots\right) \tag{8}$$

where the term in the parenthesis is the fixed-point approximation of $\log_2(e)$. This can be viewed as a bit-shift approximation as all operations involve multiplication by powers of two. In particular, the most accurate approximations of this form are

$$\Delta_{+}(d) \approx \mathbb{BS}\left(1, -d\right) \tag{9a}$$

$$\Delta_{-}(d) \approx -\mathbb{BS}\left(1.5, -d\right) \tag{9b}$$

where $\mathbb{BS}\left(a, b\right) = a2^b$ corresponds to a bit-shift in the binary representation of a by b positions to the left. The bit-shift approximations in (9) are shown in Fig. 1. Finally, note that these bit-shift approximations are equivalent to a LUT with $r = 1$ and dynamic range set by the maximum value of d possible with the bit-width of the fixed-point representation.

4. LOG-DOMAIN DNN TRAINING

Much of the computation associated with the feedforward and backpropagation operations are based on matrix multiplication. These can be implemented directly using the operations in Sections 2-3

$$z_i = \sum_j w_{ij} x_j + b_i \longleftrightarrow \underline{Z}_i = \mathop{\boxplus}_{j} \underline{W}_{ij} \boxdot \underline{X}_j \boxplus \underline{B}_i \tag{10}$$

In this section we describe log-domain versions of the other significant operations in the training of a DNN. While the general approach is applicable to all types of neural networks, we focus on multi-layer perceptrons (MLPs) to demonstrate the concept. Specifically, these are: (i) activation functions, (ii) weight initialization, (iii) soft-max operations, and (iv) dataset conversion. We also briefly discuss issues with fixed-point representation.

Activation Functions: It is most efficient to determine the log-domain equivalent of the desired activation function and implement that directly in the log domain processing. In the numerical results that follow, a leaky-ReLU [21] activation is used. This translates to a *log-leaky ReLU* (llReLU) in the log-domain

$$g_{\text{llReLU}}\left((X, s_x) \,|\beta\right) = \begin{cases} (X, s_x) & s_x = 1 \\ (X + \beta, s_x) & s_x = 0 \end{cases} \tag{11}$$

where β is a single hyper-parameter associated with this activation function. In back-propagation, the derivative of the activation function is required and, in this case of leaky-ReLU, the derivative is simple to implement directly in the log-domain.

Weight Initialization: Weights are conventionally initialized according to some specified distribution and it is most efficient to translate this distribution to the log-domain and initialize the log-domain weights accordingly. The probability density function used for weight initialization is typically symmetric around zero – i.e., $f_w\left(x\right) = f_w\left(-x\right)$. We consider such symmetric distributions for which the sign in log domain is Bernoulli distributed, equally likely to be 0 or 1. The distribution for $W = \log_2|w|$, can be determined using standard change of measure approaches from probability as

$$f_W\left(y\right) = 2^{y+1}\log_e(2) f_w\left(2^y\right) \tag{12}$$

Soft-max Layer: In classification tasks, it is common to use a final soft-max layer with a cross-entropy cost [1]. The soft-max and the associated gradient initialization are

$$p_{ij} = \frac{e^{a_{ij}}}{\sum_{j=1}^{N} e^{a_{ij}}} \tag{13a}$$

$$\delta_{ij} = p_{ij} - y_{ij} \tag{13b}$$

where y_{ij} is the one-hot encoded label. In the log-domain, this corresponds to

$$\log_2 p_{ij} = \left(a_{ij}\log_2 e\right) - \mathop{\boxplus}_{j=1}^{N}\left(a_{ij}\log_2 e, 1\right) \tag{14a}$$

$$\left(\log_2|\delta_{ij}|, s_{\delta_{ij}}\right) = \underline{P}_{ij} \boxminus \left(\log_2|y_{ij}|, s_{y_{ij}}\right) \tag{14b}$$

Dataset Conversion: The dataset used for training or the inputs used during inference also need to be converted to the log-domain. In the numerical results that follow, this was done with off-line pre-processing using floating point operations. In a real-time application this conversion requires computing $\log_2\left(\sum_i 2^i\right)$ and therefore could also be performed using the (approximate) operations in Sections 2-3.

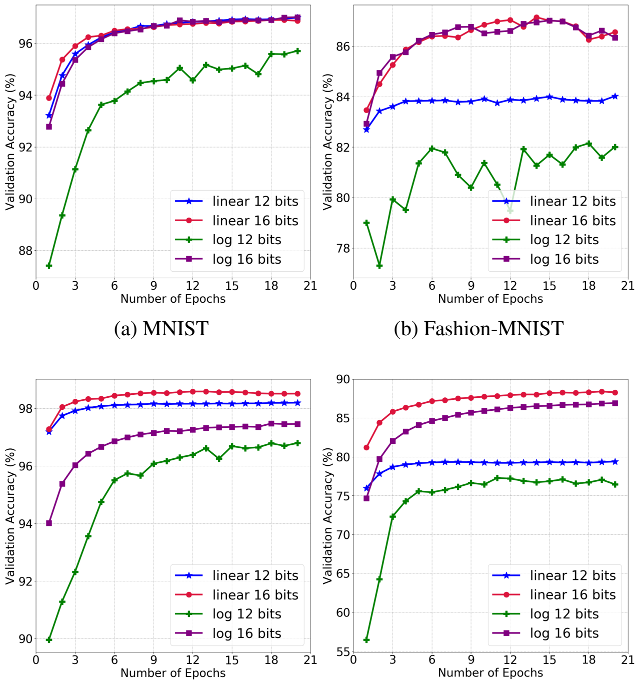
Fixed-Point Implementation: A fixed-point representation of x in the LNS described in Section 2 with q_i integer bits and q_f faction bits will have a total of $W_{\log} = 2 + q_i + q_f$ bits owing to the single bit to represent s_x and the bit for the sign of X. In analysis omitted for brevity, we show that the number of bits required in the log-domain to ensure the same range and precision as a given fixed-point representation in the linear-domain is

$$W_{\log} \geq 1 + \max\left(\lceil\log_2\left(b_i + 1\right)\rceil, \lceil\log_2 b_f\rceil\right) + W_{\text{lin}} \tag{15}$$

where W_{lin} is the bit-width in the linear domain, comprised of 1 sign bit along with b_i and b_f integer and fractional bits, respectively. For a typical value of 16-bit precision, with $b_i = 4$ and $b_f = 11$, $W_{\log} = 21$ is required to guarantee the same precision and dynamic range. The analysis leading to (15) is worst-case and our numerical experiments, summarized in Section 5, suggest that $W_{\log} \approx W_{\text{lin}}$ suffices in practice.

5. NUMERICAL EXPERIMENTS

The neural network trained is an MLP with one input layer of 784 neurons, one hidden layer of 100 neurons, and one soft-max layer with number of neurons equal to the number of classes for the given dataset. Stochastic gradient descent was used with mini-batch size of 5 and learning rate of 0.01. The weight decay regularization constant was optimized for each individual dataset. In general, 12-bit implementation needed a larger regularization constant than the 16-bit implementations. The activation function used in the hidden layer for the linear baselines is leaky-ReLU [21] and llReLU for the log experiments. When approximating Δ_{-}, its value at 0 is set to be the most negative number the fixed point setting can represent. Our experimental results are available online at [22],



(a) MNIST

(b) Fashion-MNIST

(c) EMNIST-digits

(d) EMNIST-letters

Fig. 2: Validation accuracy learning curves comparing 12 and 16 bit log-domain training with 12 and 16-bit linear training. Log-training was performed using a 20 element table ($d_{\max} = 10$, $r = 1/2$) for all operations except the soft-max which employed a 640 element table ($d_{\max} = 10$, $r = 1/64$).

where the log-domain core has been implemented in C with a Python callable wrapper.

Four balanced datasets were used for experiments: MNIST [23], Fashion-MNIST [24] (FMNIST), EMNIST-Digits (EMNISTD), and EMNIST-Letters (EMNISTL) [25]. All datasets contain 8-bit encoded, gray-scale square images with 784 pixels. MNIST and FMNIST each have 10 output classes and comprise $6,000$ training images and $1,000$ test images per class. EMNISTD has $24,000$ and $4,000$ images per each of its 10 classes for training and test, respectively. EMNISTL has $4,800$ and 800 images per each of its 26 classes for training and test, respectively. Validation data was held back from the training datasets with a 1:5 ratio.

Datasets	Float	**Linear-domain** fixed-point		**Log-domain** fixed-point look-up tables		**Log-domain** fixed-point bit-shifts	
		12b	16b	12b	16b	12b	16b
MNIST	97.4	97.3	96.9	96.0	97.2	95.5	96.5
FMNIST	87.1	82.8	88.0	80.5	87.1	79.3	85.7
EMNISTD	98.6	98.3	98.7	96.9	97.5	96.2	97.4
EMNISTL	88.1	79.7	88.7	76.4	86.7	73.7	82.5

Table 1: Test set accuracy (%) at 20 epochs. Fixed-point log-domain results are for LUT and bit-shift approximations to log-domain adds (i.e., $\Delta_{\pm}(\cdot)$ approximations).

Learning curves are shown in Figure 2 for finite precision linear and log-domain training. In the linear-domain, using 16-bit fixed-point representation, with 11 bits for the fractional part, was found to provide negligible degradation relative to floating point. In the log-domain, 16-bit representations use 10 fractional bits (i.e., owing to the extra bit needed for the sign). When performing experiments on 12 bit systems, the number of fractional bits is kept at 7 and 6 for linear and log-domain, respectively. For the LUT-based approximations in the log-domain, we first minimized the table sizes need empirically. First, high-resolution was used and the minimum value of dynamic range required for good overall accuracy was determined to be $d_{\max} = 10$. Next, fixing the dynamic range to 10, we varied the resolution and determined that $r = 1/2$ was required to achieve minimal degradation relative to linear-domain results. We found that the log-domain implementation of the soft-max was more sensitive to approximation errors and the results in Figure 2 utilize a resolution of $r = 1/64$ for the soft-max processing. Table 1 provides a comparison of test-set accuracy for fixed-point linear processing and full log-domain training and inference with various approximations.

6. CONCLUSIONS

Our results demonstrate that all training and inference processing associated with a neural network can be performed using logarithmic number system with approximate log-domain additions, thus allowing a hardware implementation without multipliers. In particular, approximating the log-domain addition using a $\max(\cdot)$, add, and an approximation to the Δ-term based on a LUT yields only modest degradation in classification accuracy as compared to that of linear processing. Similar to linear processing, we conclude that 16-bit fixed-point representations are sufficient to approach the classification accuracy associated with floating point computations. We also found that a LUT of size 20 was sufficient and that a simple bit-shift approximation, which can be viewed as equivalent to a smaller table, also provides good classification performance in many cases.

Future areas of research include application to larger convolutional neural networks popular in computer vision and co-optimization of Δ-term approximations considering classification accuracy and hardware complexity. While this work demonstrates the potential of this approach, the circuit implementation complexity of the approximate log-domain adder must be significantly lower than that of a multiplier in order for the approach to be desirable in practice. This comparison is a natural direction for future work.

7. REFERENCES

[1] Ian Goodfellow, Yoshua Bengio, and Aaron Courville, *Deep learning*, MIT press, 2016.

[2] Xiaolong Ma, Fu-Ming Guo, Wei Niu, Xue Lin, Jian Tang, Kaisheng Ma, Bin Ren, and Yanzhi Wang, "PCONV: The missing but desirable sparsity in DNN weight pruning for real-time execution on mobile devices," *http://arxiv.org/abs/1909.05073*, 2019.

[3] S. Dey, K. Huang, P. A. Beerel, and K. M. Chugg, "Predefined sparse neural networks with hardware acceleration," *IEEE Journal on Emerging and Selected Topics in Circuits and Systems*, 2019.

[4] Mahdi Nazemi, Ghasem Pasandi, and Massoud Pedram, "Nullanet: Training deep neural networks for reduced-memory-access inference," *CoRR*, vol. abs/1807.08716, 2018.

[5] Sourya Dey, Yinan Shao, Keith M. Chugg, and Peter A. Beerel, "Accelerating training of deep neural networks via sparse edge processing," in *Artificial Neural Networks and Machine Learning – ICANN 2017*, 2017.

[6] Souvik Kundu, Saurav Prakash, Haleh Akrami, Peter A. Beerel, and Keith M. Chugg, "A Pre-defined Sparse Kernel Based Convolution for Deep CNNs," *http://arxiv.org/abs/1910.00724*, 2019.

[7] Aojun Zhou, Anbang Yao, Yiwen Guo, Lin Xu, and Yurong Chen, "Incremental network quantization: Towards lossless cnns with low-precision weights," *ArXiv preprint arXiv:1702.03044*, 2017.

[8] S. C. Lee and A. D. Edgar, "Addendum to 'the focus number system'," *IEEE Transactions on Computers*, 1979.

[9] E. E. Swartzlander and A. G. Alexopoulos, "The sign/logarithm number system," *IEEE Transactions on Computers*, 1975.

[10] Patrick Robertson, Emmanuelle Villebrun, Peter Hoeher, et al., "A comparison of optimal and sub-optimal map decoding algorithms operating in the log domain," in *IEEE International Conference on Communications*, 1995.

[11] R. C. Ismail and J. N. Coleman, "ROM-less LNS," in *2011 IEEE 20th Symposium on Computer Arithmetic*, 2011.

[12] H. Fu, O. Mencer, and W. Luk, "Comparing floating-point and logarithmic number representations for reconfigurable acceleration," in *2006 IEEE International Conference on Field Programmable Technology*, 2006.

[13] N. G. Kingsbury and P. J. W. Rayner, "Digital filtering using logarithmic arithmetic," *Electronics Letters*, 1971.

[14] M. G. Arnold, T. A. Bailey, J. J. Cupal, and M. D. Winkel, "On the cost effectiveness of logarithmic arithmetic for backpropagation training on SIMD processors," in *Proceedings of International Conference on Neural Networks (ICNN'97)*, 1997.

[15] M. Arnold, J. Cowles T. Bailey, and J. Cupal, "Implementing back propagation neural nets with logarithmic arithmetic," *International AMSE conference on Neural Nets, San Diego*, 1991.

[16] E. H. Lee, D. Miyashita, E. Chai, B. Murmann, and S. S. Wong, "Lognet: Energy-efficient neural networks using logarithmic computation," in *2017 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, March 2017, pp. 5900–5904.

[17] Daisuke Miyashita, Edward H. Lee, and Boris Murmann, "Convolutional neural networks using logarithmic data representation," *http://arxiv.org/abs/1603.01025*, 2016.

[18] Jeff Johnson, "Rethinking floating point for deep learning," *CoRR*, vol. abs/1811.01721, 2018.

[19] John Gustafson and Isaac Yonemoto, "Beating floating point at its own game: Posit arithmetic," *Supercomputing Frontiers and Innovations*, vol. 4, no. 2, 2017.

[20] Ioannis Kouretas and Vassilis Paliouras, "Logarithmic number system for deep learning," in *2018 7th International Conference on Modern Circuits and Systems Technologies (MOCAST)*. IEEE, 2018, pp. 1–4.

[21] K. He, X. Zhang, S. Ren, and J. Sun, "Delving deep into rectifiers: Surpassing human-level performance on imagenet classification," in *2015 IEEE International Conference on Computer Vision (ICCV)*, 2015.

[22] "Deep Neural Networks multi-layer perceptron implementation using Logarithmic Number System," *https://github.com/arnabsanyal/lnsdnn.git*.

[23] Yann LeCun, Léon Bottou, Yoshua Bengio, Patrick Haffner, et al., "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, 1998.

[24] Han Xiao, Kashif Rasul, and Roland Vollgraf, "Fashion-mnist: a novel image dataset for benchmarking machine learning algorithms," *arXiv preprint arXiv:1708.07747*, 2017.

[25] Gregory Cohen, Saeed Afshar, Jonathan Tapson, and André van Schaik, "EMNIST: an extension of MNIST to handwritten letters," *arXiv preprint arXiv:1702.05373*, 2017.