A Lightweight Transformer Model using Neural ODE for FPGAs

Ikumi Okubo*, Keisuke Sugiura*, Hiroki Kawakami*, Hiroki Matsutani*
*Keio University, 3-14-1 Hiyoshi, Kohoku-ku, Yokohama, Japan 223-8522
Email: {okubo,sugiura,kawakami,matutani}@arc.ics.keio.ac.jp

***Abstract*—A transformer is an emerging neural network model that employs an attention mechanism. It has been adopted to various tasks and achieved a favorable accuracy compared to CNNs (Convolutional Neural Networks) and RNNs (Recurrent Neural Networks). Although the attention mechanism is recognized as a general-purpose component, many of the transformer models require a significant number of parameters and thus they are not suited to low-cost edge devices. Recently, a resource-efficient hybrid model that uses ResNet as a backbone architecture and replaces a part of its convolutional layers with an MHSA (Multi-Head Self-Attention) mechanism was proposed. In this paper, we significantly reduce the parameter size of this approach by using Neural ODE as a backbone architecture for the MHSA mechanism. The proposed hybrid model reduces the parameter size by 97.3% compared to the original model without degrading the accuracy. Since the model size is quite small, it is implemented on Xilinx ZCU104 FPGA (Field Programmable Gate Array) board so that it can fully exploit on-chip BRAM/URAM resources. The FPGA implementation is evaluated in terms of resource utilization, accuracy, performance, and power consumption. The results demonstrate that it speeds up the model by up to 2.63 times compared to a software execution without accuracy degradation.**

I. Introduction

Transformer-based [1] model architecture with a sophisticated attention mechanism, e.g., Multi-Head Self-Attention (MHSA), is currently being intensively investigated. ViT (Vision Transformer) [2] is the first pure Transformer-based architecture for visual tasks and achieves promising results compared to existing CNN-based models. However, such results are obtained only when using large datasets such as JFT-300M [3], which limits the application on limited computing resources. Generally speaking, that is because the attention mechanism lacks some of the inductive biases, e.g. locality and translational invariance. Thus, one of the solutions for alleviating the dependence on large datasets is to combine both convolution and attention mechanism [4], [5], [6], [7], [8], [9]. Such hybrid models can achieve state-of-the-art accuracy for small or middle datasets in the variety of tasks.

Among these models, we focus on BoTNet (Bottleneck Transformer) [7], which replaces the spatial convolutions with global self-attention in the last three bottleneck blocks of ResNet50 and improves upon the baselines significantly on instance segmentation and object detection tasks. Simple adoption of BoTNet design for image classification achieves 84.7% top-1 accuracy on the ImageNet benchmark.

In this paper, we propose one of the smallest Transformer-based model for resource-constrained edge devices by approximating the ResNet backbone based on a concept of Neural ODE [10]. Neural ODE (Ordinary Differential Equation) considers the discrete update law of ResNet as a continuous ordinary differential equation and calculates the numerical solution by the ODE solvers. The number of parameters is greatly reduced because the multiple building blocks in ResNet are replaced with multiple iterations of a single building block in Neural ODE.

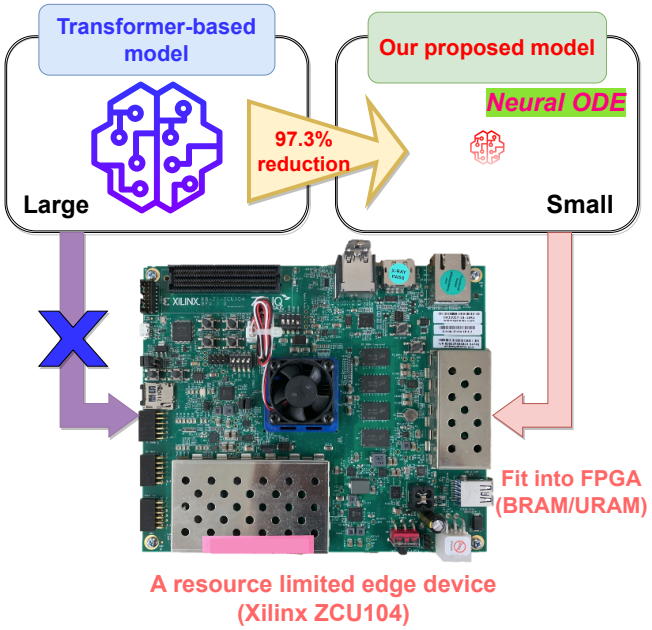

Fig. 1. Transformer-based model on FPGA

Our contribution is threefold:

(1) MHSA mechanism tends to be a computational bottleneck on the model because of its high computational complexity. Thus, we implement MHSA on FPGA (Field-Programmable Gate Array) taking the trade-off between resource utilization and performance gain into account. We can accelerate it by up to **2.63** times compared to pure software execution.

(2) BoTNet is so large that implementing on resource-limited edge devices such as Xilinx ZCU104 is not feasible. We therefore focus on Neural ODE to reduce the number of parameters in BoTNet50. The proposed model succeeds in reducing the number of parameters by **97.3**% while maintaining an accuracy comparable to the original BoTNet in the STL10 dataset.

(3) We evaluate the FPGA resource utilization, execution time, and power consumption of MHSA on the Xilinx ZCU104 board.

The rest of this paper is organized as follows. Sec. II presents related works and compares with our work. Sec. III provides the background on MHSA mechanism and Neural ODE. Sec. IV presents the design of the proposed model. Sec. V implements the proposed model on the FPGA board to accelerate the MHSA which is a computational bottleneck on the model. Sec. VI shows the evaluation results in terms of parameter size, accuracy, and resource utilization of MHSA on the FPGA board. Sec. VII concludes this paper and discusses our future work.

II. RELATED WORKS

A. Convolution and MHSA

CNNs have inductive biases such as locality and translational invariance, while MHSA captures the interrelationships among global features. ViT [2] is a model which applies pure Transformer to image recognition and achieves greater accuracy than CNN-based models under a large dataset (e.g. JFT-300M). However, one major drawback of ViT is that it has worse performance than CNN-based models when trained on small to middle datasets such as CIFAR10/100 [11] and ImageNet [12] because of some lack of inductive biases. Therefore, hybrid models that combine MHSA (global self-attention but data hungry) and convolution (having inductive biases) have been proposed [4], [5], [6], [7], [8], [9]. BoTNet [7] is introduced as a simple modification to ResNet, which replaces convolutions with MHSAs, and has shown to improve accuracy in object detection, instance segmentation, and image classification. Benefits of including MHSA in the model are also demonstrated in [7] by improving the accuracy under the condition of larger image size and scale jitter, since MHSA is capable of capturing global relationships on the contrary to lack of some inductive biases. By analyzing the relationship between convolution and MHSA, it is shown that CNN tends to increase the variance of the feature map while MHSA tends to decrease it [8]. As benefits from both mechanisms, AlterNet is proposed in [8] to suppress the dispersion of feature maps by adding MHSA to the final layer of each stage in ResNet, where dispersion peaks. It is shown that AlterNet outperforms existing models on small datasets. Furthermore, MHSA not only contributes to improved accuracy, but also to the flat and smooth loss surface, thereby increasing the model's robustness.

B. Reduction of model computational complexity

In the context of widespread application of AI on IoT devices, running the inference of large-scale networks on such devices is intractable due to device resource limitations. Therefore, various types of model have been proposed to reduce the number of parameters while minimizing accuracy loss.

ODENet [10] is a model that generalizes ResNet using neural ordinary differential equations (Neural ODEs), and the number of parameters is reduced by turning a set of building blocks into multiple iterations of a single block, i.e., reusing the same parameters. This method is also used in this paper, and a detailed explanation is given in Sec. III-B.

For a specific problem with MHSA, the computational complexity is proportional to the square of the feature map size. Many methods have been proposed to solve this. For example, Linear Transformer [13] is a kernel-based method, Transformer-XL [14] is a recursion-based method, Linformer [15] is a low-rank factorization-based method, Reformer [16] is a learnable-pattern-based method, and Swim Transformer [17] employs a fixed-pattern and global memory-based methods.

C. Transformer implemented on FPGA

There is some research on the FPGA implementation of the Transformer to improve the inference speed. The weight pruning method for Transformer is implemented in [18]. FTRANS [19] uses a cyclic matrix-based weight representation for Transformer model for natural language processing. VAQF [20] proposes an automated method that implements a quantized ViT on a Xilinx ZCU102 FPGA to meet the required accuracy. Compared to our proposal, these models are not based on CNNs and thus suffer from the drawbacks of attention mechanisms described earlier. In addition, both of these models use FPGA boards with more resources than the one used in this paper (Xilinx ZCU104), and our proposed model is the smallest Transformer-based model to the best of our knowledge.

III. PRELIMINARIES

A. MHSA

1) Attention mechanism: The attention mechanism aims to capture the relationship between a set of query $\mathbf{Q} = [\mathbf{q}_1, \ldots, \mathbf{q}_M]^T$ and a feature map in order to identify the region of interest in a feature map for each query $\mathbf{q}_i$.

Let $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_N]^T \in \mathbb{R}^{N \times D}$ be a feature map $\mathbf{x}_i \in \mathbb{R}^D$ where N and D denote a size of the feature map and a channel, respectively. Given a vector $\mathbf{q} \in \mathbb{R}^D$, called query, its attention $\boldsymbol{a} \in \mathbb{R}^N$ is calculated as follows:

$$[a_1, \ldots, a_N] = \mathbf{Softmax}\left(\frac{\mathbf{q}^T\mathbf{X}^T}{\sqrt{D}}\right). \tag{1}$$

The attention a_i represents how well the query $\mathbf{q}$ and the element of feature map $\mathbf{x}_i$ are related. By stacking Eq. (1), attentions for a set of queries can be calculated in the form of matrix operation as follows:

$$\mathbf{A} = \mathbf{Softmax}\left(\frac{\mathbf{QX^T}}{\sqrt{D}}\right), \mathbf{A} \in \mathbb{R}^{M \times N}. \tag{2}$$

Note that the softmax operator in Eq. (2) is applied rowwise to $\mathbf{QX}^T$.

2) Self-attention: Self-attention is a special case of the attention mechanism, where the input query $\mathbf{Q}$ is the feature map itself. Self-attention mechanism can learn how to genarate better feature maps, calculating correlation between every pair of feature maps by attention mechanism (Eq. (2)). First, query, key, and value matrices $\mathbf{Q}$, $\mathbf{K}$, $\mathbf{V}$ are computed from $\mathbf{X} \in \mathbb{R}^{N \times D}$ using three learnable weights $\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v \in \mathbb{R}^{D \times D}$ as follows.

$$\mathbf{XW}^q = \mathbf{Q} \in \mathbb{R}^{N \times D} \tag{3}$$

$$\mathbf{XW}^k = \mathbf{K} \in \mathbb{R}^{N \times D} \tag{4}$$

$$\mathbf{XW}^v = \mathbf{V} \in \mathbb{R}^{N \times D} \tag{5}$$

An attention map $\mathbf{A} \in \mathbb{R}^{N\times N}$ is computed by multiplying $\mathbf{Q}$ with $\mathbf{K}^T$ similar to Eq. (2), namely:

$$\mathbf{A} = \mathbf{Softmax}\left(\frac{\mathbf{QK^T}}{\sqrt{D}}\right), \mathbf{A} \in \mathbb{R}^{N\times N}. \tag{6}$$

For each query $\mathbf{q}_i \in \mathbb{R}^D$, its output is obtained by calculating the sum of values $\{\mathbf{v}_1, \ldots, \mathbf{v}_N\}$ weighted by their attention scores (i.e., coefficients of the i-th row vector of $\mathbf{A}$) $\mathbf{a}_i = [a_{i,1}, \ldots, a_{i,N}] \in \mathbb{R}^N$. Then, by taking an inner product between $\mathbf{A}$ and $\mathbf{V}$, an output of the self-attention is calculated as follows:

$$\mathbf{SA}(\mathbf{X}) = \mathbf{AV} = \mathbf{Softmax}\left(\frac{\mathbf{QK}^T}{\sqrt{D}}\right)\mathbf{V} \in \mathbb{R}^{N\times D}. \tag{7}$$

Unlike convolution operations, which only use a subset of input features (a local region of input features) to compute each output element, in the self-attention mechanism, all input elements will contribute to the output for each query $\mathbf{q}_i$. The self-attention mechanism thus aggregates the input feature globally and is not affected by inductive biases.

3) Positional encoding: Since the self-attention (Eq. (7)) is a function on the input set $[\mathbf{x}_1, \ldots, \mathbf{x}_N]^\top$, it causes a loss of the positional information. For example, in Eq. (2), if $\mathbf{x}_i$ and $\mathbf{x}_j$ are swapped, then their corresponding output elements, i.e., $\mathbf{a}_i$ and $\mathbf{a}_j$, are also swapped. The model with this property, called equivariance, is not appropriate for structural data such as images, since their outputs are invariant to the random permutation of image pixels. To deal with this information loss, a positional encoding is often employed with the attention mechanism. A position-specific vector $\mathbf{p}_i \in \mathbb{R}^D$ is added to the input at position i ($\mathbf{x}_i \in \mathbb{R}^D$) via the vector addition $\mathbf{x}_i + \mathbf{p}_i$ or the concatenation $[\mathbf{x}_i, \mathbf{p}_i]$. Encoding methods fall into two main categories; parameters are learnable relative position or absolute hyperparameters. As the typical example, Transformer [1] adopts the sinusoidal positional encoding which falls into a category of the absolute positional encoding and is a set of hyperparameters written as follows;

$$\mathbf{p}_i = \begin{cases} \sin\left(\frac{i}{10000^{2j/D}}\right) & (i = 2j) \\ \cos\left(\frac{i}{10000^{2j/D}}\right) & (i = 2j+1) \end{cases} \tag{8}$$

where $j = 1, \ldots, D/2$. On the other hand, it is proposed to learn the relative positional encoding via MLP [2], [17]. It is investigated whether absolute or relative position should be used as positional encoding and shown that relative position leads to better accuracy [7] . Our method thus uses a learnable relative positional encoding.

4) MHSA: MHSA is an extension of the self-attention mechanism [1], which employs multiple self-attention heads (concatenates outputs from multiple self-attention heads) to jointly learn different relationships between features. First, weights $\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v$ are partitioned along the second dimension into k sets (i.e., $\mathbf{W} = \mathbf{concat}[\mathbf{W}_1; \mathbf{W}_2; \ldots; \mathbf{W}_k]$), each of which is fed to a separate self-attention head. Let $\mathbf{SA}_i(\mathbf{X})$ be the i-th self-attention head, and the output of MHSA is represented by Eq. 9. In general, the output dimension of each head is arbitrary, but it is usually set to $D_h = \frac{D}{k}$.

$$\mathbf{MHSA}(\mathbf{X}) = [\mathbf{SA}_1(\mathbf{X}); \mathbf{SA}_2(\mathbf{X}); ...; \mathbf{SA}_k(\mathbf{X})], \quad \mathbf{MHSA}(\mathbf{X}) \in \mathbb{R}^{N\times kD_h} \tag{9}$$

B. Neural Ordinary Differential Equation (Neural ODE)

ResNet is a widely used backbone architecture especially for image classification tasks; it introduces the residual connections between CNN-based building blocks (**ResBlock**s) to address the vanishing gradient problem and allow training deeper networks with tens of layers. Neural ODE [10] is considered as a continuous generalization of ResNet, which interprets the skip connection as a discrete approximation of the ODE (ordinary differential equation). Let $f(\boldsymbol{z}_i, \theta_i)$ denote the i-th ResBlock with an input $\boldsymbol{z}_i$ and parameters θ_i. The computation of ResNet is formally written as recurrent updates of the hidden state $\boldsymbol{z}$ as follows:

$$\boldsymbol{z}_{i+1} = \boldsymbol{z}_i + f(\boldsymbol{z}_i, \theta_i), \quad i = 0, \ldots, N-1 \tag{10}$$

where N is the number of building blocks, and the first additive term corresponds to the skip connection.

By treating the layer index i as a time point and $\boldsymbol{z}_i$ as a time-dependent parameter evaluated at t_i, in the limit of infinitely many blocks (i.e., $N \to \infty$ and $\Delta t \to 0$), Eq. (10) turns into an ODE with respect to t as follows:

$$\frac{d\boldsymbol{z}}{dt} = f(\boldsymbol{z}(t), t, \theta). \tag{11}$$

The solution $\boldsymbol{z}(t_1)$ at some time point $t_1 \geq t_0$ is obtained by integrating Eq. (11):

$$\boldsymbol{z}(t_1) = \boldsymbol{z}(t_0) + \int_{t_0}^{t_1} f(\boldsymbol{z}(t), t, \theta)dt \tag{12}$$

$$= \mathrm{ODESolve}(\boldsymbol{z}(t_0), t_0, t_1, f), \tag{13}$$

where $\boldsymbol{z}(t_0)$ is an initial state (i.e., input to the ResNet) and ODESolve is an arbitrary ODE solver such as Euler and Runge-Kutta methods. In the Euler method, the time $t \in [t_0, t_1]$ is again discretized by $t_j = t_0 + jh$ with a small step $h = (t_1 - t_0)/N$, and $\boldsymbol{z}(t)$ is iteratively solved as follows:

$$\boldsymbol{z}(t_{j+1}) = \boldsymbol{z}(t_j) + hf(\boldsymbol{z}(t_j), t_j, \theta). \tag{14}$$

The observation of Eqs. (10) and (14) reveals that the forward propagation of a single ResBlock amounts to one iteration of the Euler method, i.e., by exploiting the Neural ODE formulation, C different ResBlocks can be merged into one block (**ODEBlock**) that computes Eq. (14) C times as shown in Fig. 2 (left).. As apparent in Eq. (10), C ResBlocks require C individual sets of parameters $\{\theta_1, \ldots, \theta_C\}$; compared to that, an ODEBlock reuses the same parameter θ during the C iterations; this reduces the number of parameters to $1/C$. Neural ODE approach allows to effectively build a compressed and lightweight alternative of the ResNet-based deep models, by replacing N ResBlocks with N/C ODEBlocks. Similarly to ResNets, models composed by a stack of ODEBlocks are referred to as ODENets in this paper.

IV. Design of the Proposed Network

This section describes the design of the proposed model in this paper. ResNet consists of ResBlocks which have residual connection and bottleneck architecture. The key idea of the BoTNet is to replace the convolutional layer with a 3×3 kernel with MHSA in the final three ResBlocks, referred to

MHSABlock. By applying such simple modifications, this model has achieved a better accuracy than ResNet in ImageNet benchmark in spite of reduction of parameter size [7]. As described in Sec. III-B, Neural ODE is viewed as a variant of ResNet with a significantly reduced number of parameters; this paper aims to further reduce the parameter size and improve accuracy by applying the idea of BoTNet to Neural ODE.

According to [21], the architecture of a baseline ODENet is shown in Fig. 2 (left), which consists of three building blocks (ODEBlocks) and two downsampling layers. In [21], to reduce the parameter size, the ODEBlock replaces a conventional convolution with DSC (Depthwise Separable Convolution) [22], [23] which performs convolution in the spatial and channel direction separately. The size of DSC parameter is NK^2+NM while that of conventional convolution parameter is NMK^2, where N, M, and K are the number of input channel, output channel, and kernel size, respectively. The size of parameters is thus reduced by approximately K^2 times, assuming that $N, M \gg K$.

Other modules in the ODEBlock are a batch normalization layer for stable learning and a ReLU layer. The downsampling layer halves the width and height of the feature map while it doubles the number of channels, i.e., taking in the feature map of size (C, H, W) and computing an output of size ($2C$, $H/2$, $W/2$). Refer to [21] for details and architecture of the downsampling layer. The computation of each ODEBlock is repeated C times, while each of the downsampling layer is executed only once; ODENet can be viewed as a deep network with $3C$ + 2 blocks in total except that every C ODEBlocks share the same parameters.

Since Neural ODE is regarded as an approximation of ResNet, the idea of BoTNet can also be applied to Neural ODE. The final ODEBlock is replaced as MHSA block which has a structure same as the BoTNet. The proposed structure is shown in Fig. 2 (right). As shown in Table IV, it allows a 97.3% reduction of the number of parameters, making it one of the smallest models based on Transformer architecture and well-suited to deployment on resource-limited computing platforms.

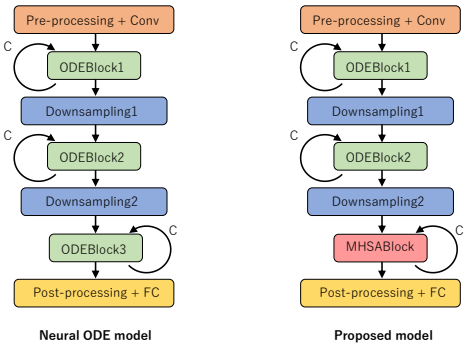

Fig. 2. **Left:** Neural ODE model (ODENet), **Right:** proposed model. The only difference is the replacement of ODEBlock with MHSABlock. MHSABlock is described in Fig. 3

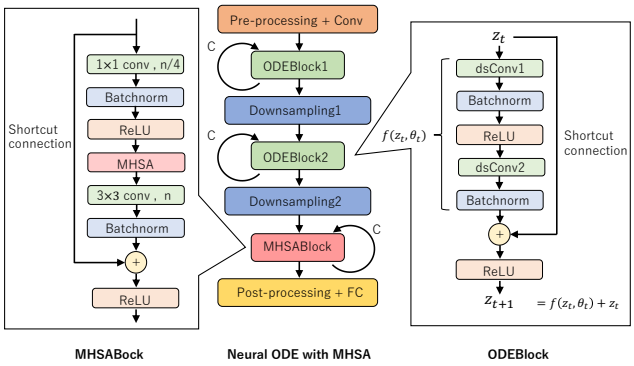

Fig. 3. BoTNet using Neural ODE

V. Implementation

In this section, we describe our proposed FPGA-based implementation of the MHSA component.

A. Modifications to MHSA

In the MHSA mechanism used in this paper, a slight modification is made to the original MHSA [1] with reference to the Bottleneck Transformer [7]. The MHSA used in this proposal is shown in Fig. 4. It uses relative positional encoding instead of absolute one as discussed in Sec. III-A. In [24], applying two-dimensional relative self-attention in column and row directions separately is better suited for vision tasks than absolute one. In Fig. 4, $\mathbf{R}_h \in \mathbb{R}^{H\times 1\times D_h}$ and $\mathbf{R}_w \in \mathbb{R}^{1\times W\times D_h}$ denote relative position vectors for vertical and horizontal directions, respectively, which are learned for each head and channel. Initial values of these vectors are drawn from a normal distribution. Let $\mathbf{1}_N \in \mathbb{R}^N$ is a vector of all ones and $\mathbf{R} = \mathbf{R}_h\mathbf{1}_N^T + \mathbf{1}_N^T\mathbf{R}_w$, then the relative position is fused into the query in the form of $\mathbf{QR}^T$. Instead of Eq. (6), the attention A is computed as follows:

$$\mathbf{A} = \mathbf{Softmax}\left(\frac{\mathbf{QK}^T + \mathbf{QR}^T}{\sqrt{D_\mathrm{h}}}\right), \mathbf{A} \in \mathbb{R}^{N\times N}. \qquad (15)$$

Another modification is to use ReLU instead of softmax as an activation function. As described in Sec. III-A, relationships between query and key (i.e., logits) are obtained by taking an inner-product of Q with K, and then they are passed on to the softmax function so that the sum of attention weights $[a_{i1}, \ldots, a_{iN}]^T$ equals to one. An immediate advantage is that ReLU is hardware-friendly as it only consumes one comparator and one multiplexer. According to [25], the accuracy of ReLU-based attention is comparable to that of the original softmax-based one, and the attention weights are sparsified, which assists the analysis of the information flow in the model. Besides, since the value of the inner product ($\mathbf{QK}^T + \mathbf{QR}^T$) is expected to be in a saturated area of the sigmoid function with vanishing gradients, LayerNorm [26] is added before the outupt of MHSA to stabilize gradients and facilitate model convergence.

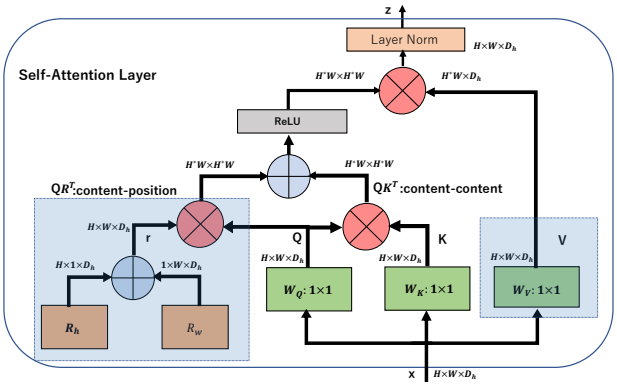

Fig. 4. Modified MHSABlock implemented in FPGA

By combining two modifications described above, the MHSA is computed as follows:

$$\mathbf{A} = \mathbf{ReLU}\left(\frac{\mathbf{QK}^T + \mathbf{QR}^T}{\sqrt{D_\mathrm{h}}}\right), \tag{16}$$

$$\begin{aligned}&\mathbf{MHSA}(\mathbf{X})\\ &= \mathbf{LayerNorm}\left([\mathbf{SA}_1(\mathbf{X}); \mathbf{SA}_2(\mathbf{X}); ...; \mathbf{SA}_k(\mathbf{X})]\right).\end{aligned} \tag{17}$$

B. Implementation of MHSA on FPGA

Here, three details of the MHSA implementation on the FPGA are shown.

1) Implementation using fixed-point: Our implementation employs fixed-point arithmetic instead of floating-point to reduce the resource consumption. The feature maps and layer inputs/outputs are in a 32-bit fixed-point format with a 16-bit integer and a 16-bit fractional part. The layer parameters are represented by a narrower 24-bit format with 8 bits and 16 bits for integer and fractional parts, respectively, considering that trained parameters are usually within a narrower value range than layer inputs/outputs and are usually concentrated around zero. The resource utilization for Xilinx ZCU104 board under these settings is shown in Table I. Compared to the floating-point version, the BRAM and DSP usages are reduced by 53% and 32%, respectively. As confirmed in Sec. VI, the fixed-point approximation does not degrade inference accuracy.

TABLE I
FPGA RESOURCES USING FLOATING POINT AND FIXED POINT

Model	BRAM	DSP	FF	LUT
Available	624	1728	460,800	230,400
512ch, 3 × 3 (floating point)	1,716(286%)	680(39%)	89,912(19%)	112,698(48%)
512ch, 3 × 3 (fixed point)	1,396(233%)	137(7%)	30,041(6%)	83,116(36%)

2) Buffer management: As shown in Eqs. (3)-(5), the input $\mathbf{X}$ is first projected into query, key, and value matrices $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$ using three learnable weights $\{\mathbf{W}^q, \mathbf{W}^k, \mathbf{W}^v\}$. The most naive approach to compute them is to first store $\mathbf{W}^q$, $\mathbf{W}^k$, $\mathbf{W}^v$, and X on four separate (individual) buffers, compute $\mathbf{Q}$, $\mathbf{K}$, and $\mathbf{V}$ matrices, and store them on three new buffers. This requires seven individual buffers in total and leads to a lower resource efficiency. Since the three matrices $\mathbf{W}^q$, $\mathbf{W}^k$, and $\mathbf{W}^v \in \mathbb{R}^{D\times D}$ account for the largest part of the BRAM usage and are of the same size, our implementation reuses a single buffer three times to compute $\{\mathbf{Q}, \mathbf{K}, \mathbf{V}\}$ as follows. It first reads $\mathbf{W}^q$ from DDR memory into the shared parameter buffer to compute $\mathbf{Q} = \mathbf{XW}^q$, and then successively overwrites the buffer with $\mathbf{W}^k$ and $\mathbf{W}^v$ to compute $\mathbf{K} = \mathbf{XW}^k$ and $\mathbf{V} = \mathbf{XW}^v$. This approach only requires five buffers in total, resulting in the reduction of total BRAM consumption by 144% as shown in Table II and thus allowing the implementation on the Xilinx ZCU104 board.

TABLE II
FPGA RESOURCES BEFORE/AFTER BUFFER MANAGEMENT

Model	BRAM	DSP	FF	LUT
Available	624	1,728	460,800	230,400
512ch, 3 × 3 before	1,396(233%)	137(7%)	30,041(6%)	83,116(36%)
512ch, 3 × 3 after	559(89%)	137(7%)	37,333(8%)	55,842(24%)

3) Parallelizing bottleneck of calculation in MHSA: The three matrix products $\mathbf{XW}^q$, $\mathbf{XW}^k$, and $\mathbf{XW}^v$ are parallelized in our method as they form the major performance bottleneck in the MHSA computation; as shown in Table III, 99% of the execution time is consumed by these large matrix products of size $(D_h, H \times W)$. In our implementation, the buffers for $\mathbf{X}$ and $\mathbf{W}$ are partitioned into 64 sub-buffers, and the innermost loop is unrolled with a factor of 128, leading to the 127x performance improvement of the matrix products and 52x overall speedup as shown in Table III.

TABLE III
PARALLELIZING COMPUTATIONAL BOTTLENECK IN MHSA

	Original		Parallelized	
Processing	Cycles	Latency (ns)	Cycles	Latency (ns)
$\mathrm{XW^q, XW^k, XW^v}$	**40,158,722**	$\mathbf{2.01 \times 10^8}$	**316,009**	$\mathbf{1.58 \times 10^6}$
$\mathrm{QR^T}$	74,132	3.71×10^5	74,132	3.71×10^5
$\mathrm{QK^T}$	78,740	3.94×10^5	78,740	3.94×10^5
$\mathrm{ReLU(QR^T + QK^T)}$	1,701	8.51×10^3	1,701	8.51×10^3
$\mathrm{ReLU(.)V^T}$	370,696	1.85×10^6	370,696	1.85×10^6
Total	**121,866,093**	$\mathbf{6.09 \times 10^8}$	**2,337,954**	$\mathbf{1.17 \times 10^7}$

C. Board-level implementation

The proposed HW/SW co-design of FPGA-based MHSA accelerator is shown in Fig. 5. In the Xilinx Zynq SoC family, HW corresponds to the PL (Programmable Logic) part, on which an MHSA IP core and a DMA (Direct Memory Access) controller are implemented. A DMA controller allows data to be directly transmitted to the MHSA IP core from DDR memory without involving the CPU. The PS (Processing System) part is for instructing the MHSA IP core, triggering the DMA controller, and computing the other part of the models (e.g., pre- and post-processing blocks in Fig. 3) for inference. Data such as parameters (e.g., weight and bias), inputs, and outputs of MHSA are transferred by using the 32-bit wide high-performance slave port (HP0 port) and utilizing AXI4-Stream protocol (red lines in Fig. 5). The control registers are connected to the high-performance master (HPM0) port via an AXI interconnect and are accessed by PS using AXI-Lite protocol and memory-mapped I/O.

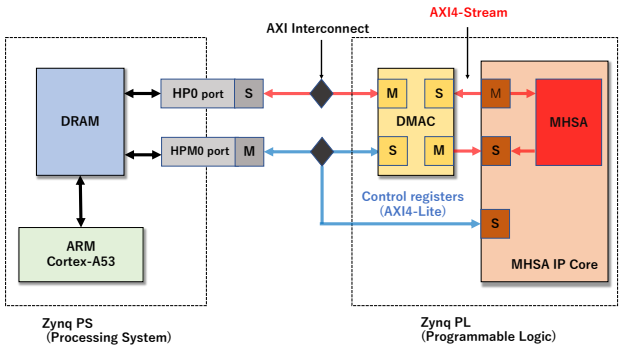

Fig. 5. Board level implementation (Xilinx Zynq UltraScale+ MPSoC)

VI. Evaluations

In this section, three models including the proposed one, BoTNet, and ViT [2] are evaluated in terms of accuracy and the number of parameters. The proposed FPGA implementations of MHSA are then evaluated in terms of performance and resource utilization.

A. Evaluation environment

1) STL10 dataset: We use labeled images of STL10 [27] dataset to evaluate the proposed model in terms of accuracy. The image size is 96×96, and the number of classes is ten. We use 5000 images for training and 8000 images for testing.

Although CIFAR-10/100 [11] datasets are widely used for the image classification tasks, their image size is relatively small (e.g., 32×32). It is expected that the attention mechanism that can aggregate global information is beneficial for larger images, and thus we use STL10 dataset in this paper. Please note that the proposed model is a combination of CNN and attention mechanism. As discussed in Sec. II-A, it is known that such a hybrid model can achieve a high accuracy even with small datasets. This is because an attention-based model (e.g., ViT) requires more samples to achieve a higher accuracy compared to CNNs where inductive biases are imposed [2]. To demonstrate the benefit of the proposed hybrid model, it is compared to ViT-Base [2] with STL10 dataset containing relatively small number of training samples.

2) Model training: The proposed model and counterpart models are implemented with Python 3.8.10 and Pytorch 1.12.1. The backbone architecture is based on [21]. As a counterpart, we use an implementation of BoTNet [1]. As an attention-based counterpart, ViT-Base is also compared to the proposed model. In this paper, the conditions of training are aligned as much as possible to maintain fairness among these models (only batch size is different). The batch size is 128 for counterpart models and 5 for the proposed model. The number of epochs is 310 and SGD is used as an optimizer. The regularization parameter of weight decay is 10^{-4}, and the momentum parameter is 0.9. CosineAnnealingWarmRestarts is used as a learning rate scheduling. The initial learning rate is 0.1, the number of initial iterations is 10, the increase factor at every restart is 2, and the minimum learning rate is 10^{-4}. As data augmentation techniques, RandomHorizontalFlip, ColorJitter, and RandomErasing implemented in the transforms library of Pytorch are used for the proposed model.

[1]https://github.com/leaderj1001/BottleneckTransformers

B. Evaluation results

1) Parameter size: Table IV shows parameter sizes of the proposed and counterpart models with and without MHSA mechanism. In BoTNet, a convolutional layer of the last stage of ResNet50 is replaced with MHSA. Similarly, in the proposed model, the last convolutional layer of Neural ODE is replaced with MHSA. As another counterpart, we implemented a ViT-Base refering to the original ViT [2]. It uses only attention mechanism without convolutional layers.

TABLE IV
Parameter size of proposed and counterpart models

Model	Number of parameters
ResNet50	23,522,362
BoTNet50	**18,885,962**
Neural ODE	599,309
Proposed model	**513,275**
ViT-Base	78,218,506

As shown in the table, BoTNet and the proposed model reduce the parameter size compared to their original models (i.e., ResNet50 and Neural ODE) by replacing the last convolutional layer with MHSA. Specifically, BoTNet reduces the parameter size by 19.7% compared to ResNet50. The proposed model reduces the parameter size by **97.3**% compared to BoTNet by using Neural ODE as a backbone architecture. In comparison with the above, the ViT-Base is quite large compared to the others.

2) Accuracy: Table V shows test accuracies of the proposed and counterpart models with and without MHSA mechanism. STL10 dataset is used for the evaluation.

TABLE V
Accuracy of proposed and counterpart models

Model	Accuracy (%)
ResNet50	79.20
BoTNet	81.60 (**+2.40**)
Neural ODE	79.81
Proposed model	80.01 (**+0.20**)
ViT-Base	62.59

As shown in Table V and combined with the results in Table IV, accuracies of BoTNet and the proposed model are comparable or higher than those of their original models, despite the reduced number of parameters by introducing MHSA. The accuracy of the ViT-Base is not high for STL10 dataset that has relatively small number of training samples. This is because attention-based models without convolutional layers require more samples to achieve a higher accuracy compared to CNNs where inductive biases are imposed, as discussed earlier. Conversely, it is a benefit of the hybrid models (i.e., BoTNet and the proposed model) when they are used in edge environments without plenty of training data.

Figs. 6-8 show test accuracies (i.e., accuracy vs. epochs) of BoTNet, the proposed model, and ViT-Base, respectively. In any cases, similar learning curves obtained, resulting that

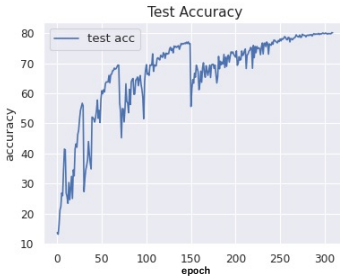

Fig. 6. Test accuracy of BoTNet

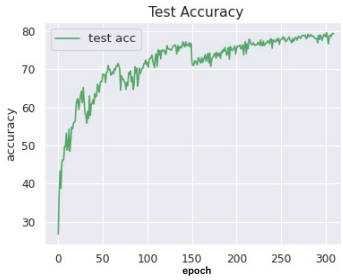

Fig. 7. Test accuracy of proposed model

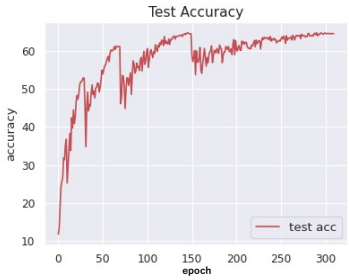

Fig. 8. Test accuracy of ViT-Base

learning has converged for all the models. Please note that the reason why the curves do not appear monotonically increasing is due to the learning rate scheduler (details are mentioned in Sec. VI-A2).

3) Execution time ratio of MHSA: Since it is assumed that computational complexity of MHSA mechanism is higher than those of the other modules, we evaluate the execution time ratio of the MHSA mechanism. Table VI shows the execution time ratios of the MHSA mechanism in MHSABlock of BoTNet and the proposed model when they are executed as software.

As shown in the table, the execution time ratios of the MHSA mechanism account for a large portion in the proposed model. Thus, accelerating the MHSA mechanism can speed up the overall models. Since the MHSA mechanism can be used as one of general components, it is also beneficial for the other emerging MHSA-based models. In this paper, the computational bottleneck of the MHSA mechanism is accelerated by using FPGA. The next section shows the evaluation results.

TABLE VI
EXECUTION TIME RATIO OF MHSA IN MHSABLOCK (%)

Model	Execution time ratio of MHSA
BoTNet	20.5
Proposed model	50.7

4) FPGA resource utilization of MHSA: The proposed model is an optimized version that reduces the parameter size of the MHSA mechanism. We implemented the MHSABlocks of different number representations (fixed and floating-point) and different sizes ((512,3,3) and (64,6,6)). The FPGA implementation is evaluated in terms of the execution time, resource utilization, and power consumption.

Table VII shows the FPGA resource utilization. In this evaluation, URAMs (Ultra RAMs) are not used in the FPGA implantation so that the BRAM utilization sharply reflects the model size differences. Please note that the floating-point arithmetic version of BoTNet can be implemented on the target FPGA if URAMs are used.

As shown in the table, resource utilizations of DSP, FFT, and LUT are significantly reduced by using the fixed-point number representations compared to the floating-point arithmetic versions.

TABLE VII
FPGA RESOURCE UTILIZATION OF MHSA OF PROPOSED MODEL

Model	BRAM	DSP	FF	LUT
Available	624	1,728	460,800	230,400
512ch, 3×3 (floating-point)	693 (111%)	680 (39%)	101,851 (22%)	90,072 (39%)
512ch, 3×3 (fixed-point)	559 (89%)	137 (7%)	37,333 (8%)	55,842 (24%)
256ch, 6×6 (floating-point)	441 (70%)	868 (50%)	144,263 (31%)	124,091 (53%)
256ch, 6×6 (fixed-point)	433 (69%)	212 (12%)	68,809 (14%)	79,476 (34%)

5) Accuracy of FPGA implementation: In the FPGA implementation, fixed-point number representations are used in feature maps and weight parameters to reduce the FPGA resource utilization. Here, we evaluate the tradeoffs between accuracy and bit widths of the fixed-point representations.

The fixed-point number implementations are denoted as $F_{total}(F_{int})-P_{total}(P_{int})$, where F_{total} and F_{int} are the numbers of total bits and integer bits for feature maps and input images, and P_{total} and P_{int} are those for weight parameters. We evaluate the inference accuracy of the following implementations: 32(16)-24(8), 24(12)-20(6), 20(10)-16(4), 18(9)-14(4), and 16(8)-12(4). In this evaluation, the MHSABlock is computed by PL part and the other parts are computed by PS part of the ZCU104 board.

Table VIII shows their test accuracy. Please note that the numerical error due to the quantization directly appears at the input values to the final FC layer rather than the classification results. The input values to the final FC layer of the FPGA implementation are compared to those of the software implementation. Figs. 9 and 10 show the mean and maximum differences between these values of the FPGA and software implementations. As shown in these tables, no accuracy degradation is observed in the FPGA implementations of 32(16)-24(8) and 24(12)-20(6).

TABLE VIII
ACCURACY VS. FIXED-POINT REPRESENTATIONS

Model	Accuracy (%)
Original	78.7
Floating-point number	78.7 (**No degradation**)
32(16)-24(8)	78.7 (**No degradation**)
24(12)-20(6)	78.7 (**No degradation**)
20(10)-16(4)	76.9 (**-0.18**)
18(9)-14(4)	59.8 (**-18.9**)
16(8)-12(4)	16.9 (**-61.8**)

6) Execution time: Table IX shows execution times of the software and FPGA implementations. As shown in the table,

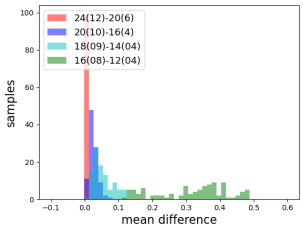

Fig. 9. Mean value difference between software and FPGA implementations

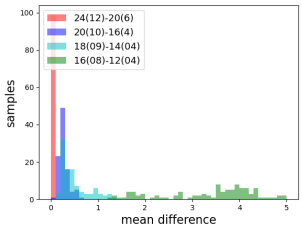

Fig. 10. Maximum value difference between software and FPGA implementations

the FPGA implementation using floating-point numbers and that using fixed-point numbers are 1.45 times and 2.63 times faster than the software implementation, respectively.

TABLE IX
EXECUTION TIME OF CPU AND FPGA IMPLEMENTATIONS (MSEC)

Model	Mean	Max	Standard deviation
CPU	35.18	36.24	0.20
FPGA (floating-point)	24.21	24.78	0.07
FPGA (fixed-point)	13.37	14.49	0.13

7) Power consumption: Power consumption of the MHSA IP core is 0.866W when the fixed-point number representation is used. It is 3.977W when the floating-point number representation is used. Power consumption of CPU (PS part of Zynq) is 2.647W. The FPGA implementation using fixed-point numbers consumes 1.33 times more power consumption while it accelerates the execution time by 2.63 times; thus, the energy efficiency is improved by 1.98 times compared to the software implementation.

VII. SUMMARY

Transformer based model, which is a trend in AI, is typically too large to implement on edge devices despite of its high accuracy. This paper focused on a recently-proposed lightweight ResNet variant, Neural ODE, and applied an idea in BoTNet by replacing the last ResNet block with an MHSA block, resulting in a model with 97.3% less parameters compared to ResNet. Furthermore, MHSA has been incorporated into various models in recent years, so this paper implemented MHSA onto FPGA fabric and evaluated its resource consumption, execution time, and power consumption. It also discusses the trade-off between the number representation and accuracy. As a future work, we are currently implementing the proposed model on the FPGA entirely to further improve the performance.

Acknowledgments This work was partially supported by JST AIP Acceleration Research JPMJCR23U3 and JSPS KAKENHI Grant Numbers JP22H03596, Japan.

REFERENCES

[1] A. Vaswani *et al.*, "Attention is All you Need," in *Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Jun. 2017, pp. 5998–6008.

[2] A. Dosovitskiy *et al.*, "An Image is Worth 16x16 Words: Transformers for Image Recognition at Scale," in *Proceedings of the International Conference on Learning Representations (ICLR)*, 2021.

[3] C. Sun *et al.*, "Revisiting Unreasonable Effectiveness of Data in Deep Learning Era," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2017, pp. 843–852.

[4] K. Yuan *et al.*, "Incorporating Convolution Designs Into Visual Transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2021, pp. 579–588.

[5] H. Wu *et al.*, "CvT: Introducing Convolutions to Vision Transformers," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Oct 2021, pp. 22–31.

[6] A. Trockman *et al.*, "Patches Are All You Need?" arXiv:2201.09792, jan 2022.

[7] A. Srinivas *et al.*, "Bottleneck Transformers for Visual Recognition," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Aug 2021, pp. 16 514–16 524.

[8] N. Park *et al.*, "How Do Vision Transformers Work?" in *Proceeding of the International Conference on Learning Representations (ICLR)*, Apr 2022.

[9] Z. Dai *et al.*, "CoAtNet: Marrying Convolution and Attention for All Data Sizes," arXiv:2106.04803, Jun 2021.

[10] R. T. Q. Chen *et al.*, "Neural Ordinary Differential Equations," in *in Proceedings of the Advances in Neural Information Processing Systems (NeurIPS)*, Dec 2018, pp. 6571–6583.

[11] A. Krizhevsky, "Learning multiple layers of features from tiny images," University of Toronto, Tech. Rep. 0, Apr 2009.

[12] O. Russakovsky *et al.*, "ImageNet Large Scale Visual Recognition Challenge," *International Journal of Computer Vision (IJCV)*, vol. 115(3), pp. 211–252, Dec 2015.

[13] A. Katharopoulos *et al.*, "Transformers are RNNs: fast autoregressive transformers with linear attention," in *Proceedings of the International Conference on Machine Learning (ICML)*, Jul 2020, pp. 5156–5165.

[14] Z. Dai *et al.*, "Transformer-xl: Attentive language models beyond a fixed-length context," in *Proceedings of the Annual Meeting of the Association for Computational Linguistics (ACL)*, jul 2019, pp. 2978–2988.

[15] S. Wang *et al.*, "Linformer: Self-Attention with Linear Complexity," arXiv:2006.04768, jun 2020.

[16] N. Kitaev *et al.*, "Reformer: The Efficient Transformer," in *Proceedings of the International Conference on Learning Representations (ICLR)*, Apr 2020.

[17] Z. Liu *et al.*, "Swin Transformer: Hierarchical Vision Transformer using Shifted Windows," in *Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Mar 2021, pp. 10 012–10 022.

[18] H. Peng *et al.*, "Accelerating Transformer-based Deep Learning Models on FPGAs using Column Balanced Block Pruning," Apr 2021, pp. 142–148.

[19] B. Li *et al.*, "FTRANS: Energy-Efficient Acceleration of Transformers Using FPGA," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design (ISLPED)*, Aug 2020, pp. 175–180.

[20] M. Sun *et al.*, "VAQF: Fully Automatic Software-Hardware Co-Design Framework for Low-Bit Vision Transformer," arXiv:2201.06618, Jan 2022.

[21] H. Kawakami *et al.*, "dsODENet: Neural ODE and Depthwise Separable Convolution for Domain Adaptation on FPGAs," in *Proceeding of the Euromicro International Conference on Parallel, Distributed, and Network-Based Processing (PDP)*, Mar 2022, pp. 152–156.

[22] A. G. Howard, M. Zhu, B. Chen, D. Kalenichenko, W. Wang, T. Weyand, M. Andreetto, and H. Adam, "MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications," arXiv:1704.04861, 2017.

[23] F. Chollet, "Xception: Deep Learning with Depthwise Separable Convolutions," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jul 2017, pp. 1800–1807.

[24] I. Bello *et al.*, "Atention augmented convolutional networks," in *In Proceedings of the IEEE/CVF International Conference on Computer Vision (ICCV)*, Nov 2019, pp. 3286–3295.

[25] B. Zhang *et al.*, "Sparse Attention with Linear Units," in *Proceedings of the Conference on Empirical Methods in Natural Language Processing (EMNLP)*, Nov 2021, pp. 6507–6520.

[26] ——, "Root Mean Square Layer Normalization," in *in Proceedings of the Neural Information Processing Systems (NeuralIPS)*, Oct 2019, pp. 12 360–12 371.

[27] A. Coates *et al.*, "An Analysis of Single-Layer Networks in Unsupervised Feature Learning," in *Proceedings of the International Conference on Artificial Intelligence and Statistics (AISTATS)*, Apr 2011, pp. 215–223.