KFUPM
Journal of Undergraduate Research International

Diamond Open Access 45

Article Computer Engineering/Computer Science

FPGA-Based Accelerator for Quantized CNNs: High-Throughput Edge Deployment with Optimized Resource Utilization

Zeyad Emad Abdel-Mawjoud[1] and Ahmed S. Abd-Rabou Mohammed[2*]

Cite https://doi.org/10.64589/juri/209734

Submitted: June 04, 2025 **Revised:** July 30, 2025 **Accepted:** August 20, 2025

ABSTRACT

This paper presents MaxNet, a high-throughput field-programmable gate array (FPGA)-based accelerator for quantized convolutional neural networks (CNNs) designed to meet the demand for efficient edge artificial intelligence (AI) deployment in low-cost hardware. By targeting the Intel MAX 10 Field Programmable Gate Array (FPGA) (10M08DAF484C8GES), MaxNet was distinguishable from prior works focused on high-end platforms, offering a tailored solution for resource-constrained applications such as the Internet of Things (IOT) and embedded vision. The optimized two-layer CNN with 8-bit quantization (Q0.8 inputs/activations, Q1.7 weights) achieved 77% accuracy with the CIFAR-10 database, demonstrating a throughput of 8,065 frames per second (fps; 0.124 ms/image) and power consumption of 1.2 W, using 861 lookup tables (LUTs) (11%) and 9 M9K memory blocks (2.9%), implemented, synthesized, and tested using Intel Quartus Prime Standard Edition 22.1 Comprehensive power measurements derived from Intel Quartus Power Analyzer (for FPGA design), psutil (a Python-based system monitoring library for CPU measurements), and nvidia-smi (NVIDIA's GPU management and monitoring utility)*superior efficiency compared to the central processing unit (Intel Core i5-10400, 578 frames per second (fps), 15 W) and graphics processing unit (NVIDIA GTX 1650, 730 fps, 50 W) baselines. Ablation studies validated the two-layer design and quantization choices, while sensitivity analysis was used to optimize the clock frequency and numerical formats. The resource utilization and per-class accuracy were confirmed, reinforcing MaxNet's suitability for low-power, high-performance edge AI in cost-effective FPGAs. This work is expected to pave the way for exploring 4-bit quantization, structured pruning, and transfer learning for domain-specific applications, such as medical imaging.

Keywords: deep learning inference, quantized CNN, GenAI acceleration, FPGA, edge AI, resource-constrained deployment, hardware--software co-design

1. INTRODUCTION

Convolutional neural networks (CNNs) have become the foundation of modern computer vision, achieving high accuracy in tasks such as image classification, object detection, and autonomous navigation [Sandler et al., 2018; Iandola et al., 2016]. Their ability to extract hierarchical features from raw data has enabled a wide range of real-world applications, including medical imaging, facial recognition, and industrial fault detection [Chung et al., 2023].

Despite their effectiveness, CNNs are computationally intensive, demanding substantial processing power, memory, and energy constraints that complicate deployment on embedded or portable devices. Edge artificial intelligence (edge AI) refers to performing AI computations directly on devices located at the "edge" of the network, closer to where the data is generated, rather than relying on remote cloud servers. Examples include IoT nodes, drones, and wearable systems. By reducing dependence on cloud connectivity, edge AI enables real-time, low-latency inference while operating under strict energy and memory budgets [Mazumder et al., 2022; Wang et al., 2022]. To meet these constraints, research has emphasized model compression and efficient architectures including quantization, pruning, and lightweight backbones (e.g., MobileNetV2, SqueezeNet), as well as low-precision designs and FPGA-oriented frameworks (e.g., FINN, LUTNet) [Sandler et al., 2018; Iandola et al., 2016; Umuroglu et al., 2017; Wang et al., 2019].

Field-programmable gate arrays (FPGAs), reconfigurable chips programmable after fabrication, are promising edge-AI accelerators due to their parallelism, reconfigurability, and low-power consumption compared to traditional processors [Hou et al., 2020; Cho & Kim, 2020]. High-end platforms (e.g., Xilinx Zynq and Virtex) have been widely used for CNN acceleration with high-throughput and scalability. However, low-cost FPGAs (e.g., Intel MAX 10) remain relatively underexplored because limited logic, multipliers, and on-chip memory restrict deployable model size and throughput [Wang et al., 2022].

Considering this gap, we developed MaxNet, a lightweight, 8-bit quantized CNN optimized for the Intel MAX 10 FPGA, to investigate the feasibility of high-throughput, low-power

Published under an exclusive license by KFUPM

inference under strict hardware constraints. The model was trained with quantization-aware training and evaluated on the CIFAR-10 dataset. Our contribution is a resource-efficient CNN architecture and implementation framework that advances practical edge-AI deployment on cost-effective FPGAs, with applicability to IoT, embedded vision, and wearable devices.

2. METHODOLOGY

2.1. Overview of MaxNet and the study design. MaxNet is a lightweight CNN optimized for the Intel MAX 10 FPGA (10M08DAF484C8GES, 8,064 logic elements, 387,072 memory bits, and 48 9-bit multipliers), targeting edge AI applications. The architecture comprised two convolutional layers (3×3 kernels, 16 and 32 filters), batch normalization (BN), rectification linear unit (ReLU) activation, 2×2 max-pooling, global average pooling (GAP), and a dense layer for image classification (10 classes, 60,000 32×32 RGB images). The CNN was implemented in Quartus Prime Standard software (Edition 22.1) to validate the minimalist design and compare the effect of the number of layers on performance, e.g., lookup table (LUT) usage, accuracy, and throughput. The simple two-layer architecture was designed to minimize pipeline stages and achieve ultra-low latency, thereby supporting a "micro-acceleration" approach, enabling potential multi-instance deployment for ensemble learning or multi-task processing.

2.2. CNN layer design. An overview of the MaxNet CNN layer architecture is shown in Figure 1, which illustrates the sequential flow of convolution, pooling, and classification layers. The input passes through two convolutional–pooling stages, followed by global average pooling and a fully connected dense layer leading to the final softmax output. The properties of each layer, including kernel size, number of filters, and output dimensions, are summarized in Table 1. This combination of figure and table provides both a visual representation of the overall architecture and a detailed specification of the design parameters. Each layer is further defined and discussed in detail in the subsequent sections.

- ***Input quantization***: Each input pixel $X_{h,w,c} \in [-1,1]$, where h and w denote the spatial coordinates and c denotes the color channel, was quantized to 8-bit precision using the Q0.8 fixed-point format. This process converts the continuous input into a clipped integer representation:

$$I_{h,w,c} = clip\left(\left\lfloor \frac{X_{h,w,c}}{2^{-7}} \right\rfloor, -128, 127\right) \quad (1)$$

Where $I_{h,w,c}$ is the quantized pixel value, and the function $clip(.)$ ensures that the result lies within the valid integer range $[-128,127]$.

- ***Convolutional Layer***: The convolutional stage generates feature maps by applying a set of kernels to the quantized input. The output feature value at spatial position (i,j) for the output channel k is computed as

$$Y_{i,j,k} = \sum_{c=0}^{C_{in}-1} \sum_{m=0}^{2} \sum_{n=0}^{2} W_{k,c,m,n} \cdot I_{i+m,j+n,c} + \beta_k \quad (2)$$

where C_{in} represents the number of input channels, $W_{k,c,m,n}$ denotes the convolutional kernel weight associated with output channel k, input channel c, and kernel coordinates (m,n), and $I_{i+m,j+n}$ is the corresponding input pixel value. This operation extracts local spatial patterns from the input image.

- ***Batch normalization***: To stabilize the distribution of features and accelerate training convergence, batch normalization was applied after each convolution. The standard BN transformation for output channel k is expressed as

$$\hat{Y}_{i,j,k} = \gamma_k \cdot \frac{Y_{i,j,k} - \mu_k}{\sqrt{\sigma_k^2 + \epsilon}} + \beta_k \quad (3)$$

Where μ_k and σ_k^2 denote the mean and variance of activations for channel k, γ_k is a learned scaling factor, β_k is a learned shift parameter, and ϵ is a small constant to avoid division by zero. For FPGA deployment, this expression was simplified into an equivalent linear transformation.

$$\hat{Y}_{i,j,k} = A_k Y_{i,j,k} + B_k \quad (4)$$

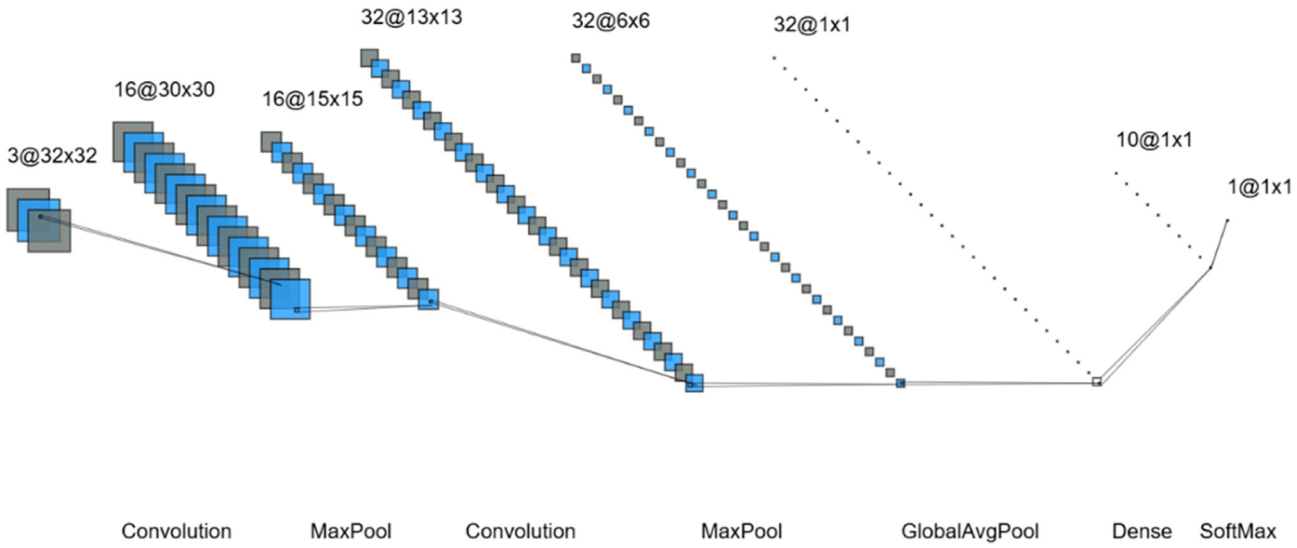

Figure 1. MaxNet CNN Architecture Diagram

where the constants

$$A_k = \frac{G_k}{\sqrt{\sigma_k^2 + \epsilon}} \quad B_k = \frac{-\mu_k}{\sqrt{\sigma_k^2 + \epsilon}} + \beta_k \tag{5}$$

They were precomputed offline. This eliminates runtime division and square-root operations, thereby improving inference efficiency.

- ***ReLU activation:*** Following batch normalization, the rectified linear unit (ReLU) activation function introduces nonlinearity into the network by suppressing negative activations. It is defined as

$$\mathrm{Z_{i,j,k}} = \max\left(0, \hat{Y}_{i,j,k}\right) \tag{6}$$

Where $Z_{i,j,k}$ represent the activated feature value. ReLU not only enables the network to learn complex relationships but also simplifies hardware implementation, as it can be realized using simple comparators.

- ***Max-pooling:*** To reduce the spatial resolution while retaining the most salient features, max pooling was applied over non-overlapping 2×2 windows. For each output location (i, j) and channel k, the pooled value is given by

$$P_{i,j,k} = max\left(Z_{2i,2j,k}, Z_{2i+1,2j,k}, Z_{2i,2j+1,k}, Z_{2i+1,2j+1,k}\right) \tag{7}$$

Where $P_{i,j}$, and k denote the reduced feature map. This downsampling operation halves the spatial dimensions, thereby reducing computational cost and memory usage in subsequent layers.

- ***Global average pooling***:Instead of employing fully connected layers that flatten the feature maps, MaxNet uses global average pooling (GAP) to reduce each feature map to a single representative value. For channel k, the GAP output is computed as

$$G_k = \frac{1}{H * W} \sum_{i=0}^{H-1} \sum_{j=0}^{W-1} P_{i,j,k} \tag{8}$$

Where H and W are the height and width of the pooled feature map. This approach greatly reduces the number of parameters, mitigates overfitting, and produces a compact representation for classification.

- ***Dense layer and output:*** The final dense layer produces the class logits by linearly combining the GAP outputs. For each class $c \in \{0, 1, \ldots, 9\}$, the output is defined as

$$O_c = \sum_{k=0}^{31} W_{c,k} \cdot G_k + \beta_c, c \in \{0, \ldots, 9\} \tag{9}$$

Where O_c is the logit corresponding to class c, $\mathrm{W_{c,k}}$ represents the weight connecting GAP feature $\mathrm{G_k}$ to class c, and β_c is the class-specific bias. All weights and biases were quantized to the same fixed-point formats as earlier layers, ensuring consistent precision throughout the pipeline.

2.3. FPGA architecture design. The MaxNet accelerator used a streaming-pipelined VHDL architecture on an Intel MAX 10 (10M08SAE144C8GES) FPGA with 8,064 LEs, 101 I/Os, 387,072 memory bits, and 48 9-bit multipliers. A custom control unit with buffers and a finite state machine (FSM) managed the data flow between layers to reduce latency. The carefully designed 4-stage pipeline breaks down the convolution computations into smaller sequential operations, allowing each convolutional output to be computed in approximately 6–12 clock cycles. This pipeline increases throughput and reduces critical path delays, enabling a clock frequency of 100 MHz.

The FPGA design includes dedicated hardware modules for convolution, BN, ReLU, max-pooling, and GAP. These blocks communicate through a streaming interface and are efficiently mapped to M9K embedded memory blocks using fixed-point representations (Q0.8, Q8.8, and Q16.16) for the intermediate data and weights. The use of dedicated read-only memory (ROM) and random access memory (RAM) units at each stage ensures minimal bottlenecks in memory access and processing.

2.4. Quantization-aware training. Quantization-aware training (QAT) was performed using the TensorFlow/Keras deep learning framework [Abadi et al., 2016; Chollet, 2015] on images from the CIFAR-10 dataset [Krizhevsky & Hinton, 2009]. The training simulated 8-bit fixed-point arithmetic, employing the Q0.8 format for inputs and activations, Q1.7 for weights, and Q16.16 for biases, where Q denotes the quantization scheme and the number indicates the allocated fractional and integer bits.

Table 1. *MaxNet Layer Overview*

Layer	InputShape	InputFormat	OutputShape	OutputFormat
Input	$32 \times 32 \times 3$	Q0.8	$32 \times 32 \times 3$	Q0.8
*Conv*1 $(3 \times 3, 16)$	$32 \times 32 \times 3$	Q0.8	$30 \times 30 \times 16$	Q16.16
*BatchNorm*1 + *ReLU*	$30 \times 30 \times 16$	Q16.16	$30 \times 30 \times 16$	Q0.8
Ma × *Pool*1	$30 \times 30 \times 16$	Q0.8	$15 \times 15 \times 16$	Q0.8
*Conv*2 $(3 \times 3, 32)$	$15 \times 15 \times 16$	Q0.8	$13 \times 13 \times 32$	Q0.8
*BatchNorm*2 + *ReLU*	$13 \times 13 \times 32$	Q0.8	$13 \times 13 \times 32$	Q0.8
Ma × *Pool*2	$13 \times 13 \times 32$	Q0.8	$6 \times 6 \times 32$	Q0.8
GlobalAvgPool	$6 \times 6 \times 32$	Q0.8	32	Q0.8
Dense	32	Q0.8	10	Q0.8

2.5. Memory architecture and allocation. Effectively managing the memory allocation is critical for achieving the efficient operation of MaxNet on the resource-constrained MAX 10 FPGA. All weights, biases, and intermediate feature maps are stored in the on-chip memory to eliminate external memory access and reduce latency and power usage. Table 2 details the allocation and usage of the FPGA's M9K memory blocks. The table also lists the data formats, sizes (in kilobytes, where 1 K =1024 bytes), and memory-block roles. Each memory block serves a specific purpose, ranging from storing weights and biases to holding intermediate feature maps and outputs. For example, the ROM blocks store the fixed parameters (weights, biases, and batch normalization coefficients) and RAM blocks store input images and intermediate feature maps. In the table, Conv1 and Conv2 are the first and second convolution layers, respectively.

2.6. Memory optimization.

- **Buffer-based data flow**: The FSM control unit uses on-chip buffers (implemented as dual-port RAM) to stage intermediate feature maps, ensuring continuous data availability for the 4-stage pipeline to avoid pipeline stalls.
- **Memory packing**: Weights and coefficients are packed efficiently (e.g., Q1.7 for Conv weights, Q1.15 for BN coefficients) to minimize the memory footprint. For example, Conv1 weights ($3 \times 3 \times 3 \times 16 = 432$ weights) use 0.422 KB, leveraging the compact representation of Q16.16.
- **No external memory**: All data resides on-chip, eliminating external Dynamic Random Access Memory (DRAM) access (common in high-end FPGAs), which reduces power and latency but constrains the model size, justifying the two-layer CNN.
- **Resource allocation**: The use of Quartus Prime Standard Edition 22.1 optimizes memory-block assignments to balance storage and logic requirements.

2.7. Control logic unit using FSM. The control logic unit employs a custom buffer-based FSM implemented in top_module.vhd to manage the data flow across the 4-stage pipeline (Conv1+BN+ReLU+MaxPool1, Conv2+BN+ReLU+MaxPool2, GAP, Dense layer). Unlike first-in-first-out (FIFO)-based designs that can introduce stalls owing to data dependencies, FSM uses on-chip dual-port RAM buffers to stage intermediate feature maps (e.g., $30 \times 30 \times 16$ after Conv1 and $6 \times 6 \times 32$ after MaxPool2). The FSM operates as follows.

- **States**: The FSM transitions through the states for each pipeline stage (e.g., LOAD_INPUT, CONV1, BN_RELU1, MAXPOOL1), coordinating data reads/writes, and computation triggers.
- **Buffer management**: Buffers (e.g., RAM_C1_Out and RAM_MF1_Out) ensure data availability, allowing parallel execution of multiply-accumulate (MAC) operations and memory access. For example, Conv1 outputs are buffered, whereas the BN + ReLU stage directly processes these outputs, thereby reducing pipeline stalls.
- **Latency reduction**: The buffer-based approach minimizes data transfer delays owing to the optimized scheduling of memory read/write functions and computation overlap.
- **Synchronization**: The FSM ensures synchronization across modules using a 100 MHz clock to trigger transitions, completing one image inference in 6–12 cycles (0.124 ms).

The efficiency of the FSM was validated using the tb_top_module.vhd test bench, which simulated the CIFAR-10 image inputs and verified the output logits.

2.8. Implementation methodology. MaxNet's FPGA implementation utilizes a fully pipelined VHDL architecture specifically designed to accommodate the resource constraints of the Intel MAX 10 FPGA. The design leverages parallelized convolution engines and a buffer-based FSM control unit to optimize data flow and reduce latency. CNNs were selected for their linear

Table 2. *Memory block specifications for MaxNet on the MAX 10 FPGA*

Memory Block	Contents	Format	Size (KB)	Purpose
ROM_W_Conv1	Conv1 weights ($3 \times 3 \times 3 \times 16$)	Q16.16	0.422	Kernel weights
ROM_W_Dense	Dense weights (32×10)	Q8.8	0.313	Kernel weights
ROM_B_Conv1	Conv1 biases (1×16)	Q16.16	0.016	-
ROM_B_Conv2	Conv2 biases (1×32)	Q8.8	0.016	-
ROM_BN1_Coeff	BN1 A/B coefficients	Q1.15	0.016	Packed A_k/B_k
RAM_InImg	Input image ($32 \times 32 \times 3$)	Q8	3	Dual-port BN_ReLU input for simultaneous read/write
RAM_C1_Out	Conv1 out ($30 \times 30 \times 16$)	Q16.16	14.4	BN-ReLU input
RAM_BR1_Out	BN+ReLU1 out ($30 \times 30 \times 16$)	Q8	14.4	Combined output
RAM_MF1_Out	MaxPool1 out ($15 \times 15 \times 16$)	Q8	3.6	Stride 2
RAM_C2_Out	Conv2 out ($13 \times 13 \times 32$)	Q8	5.4	BN-ReLU input
RAM_BR2_Out	BN+ReLU2 out ($13 \times 13 \times 32$)	Q8	5.4	Combined output
RAM_MP2_Out	MaxPool2 out ($6 \times 6 \times 32$)	Q8	1.2	Stride 2
RAM_GAP_Out	GAP (1×32)	Q8	0.031	-
RAM_Dense_Out	Dense outputs (10)	Q8	0.01	-

data flow, aligned with the 387,072-bit memory and 48 9-bit multipliers of MAX 10. The modular design ensures scalability and maintainability, and is organized into core processing modules and integration/control components.

2.8.1. Core modules.

- **conv_engine.vhd** implements convolutional operations with parameterized 3 × 3 filters and input feature maps, performing MAC operations using the 48 9-bit multipliers of MAX 10 for parallel processing. Weights were stored in the Q1.7 format, with inputs in Q0.8, and outputs in Q16.16, to maintain precision during accumulation.
- **BatchNorm_ReLU_engine.vhd** combines BN and ReLU activation to stabilize training and introduce nonlinearity. BN coefficients (A_k, B_k) are precomputed offline in Q1.15 format, and the ReLU uses comparators to zero negative values, minimizing logic overhead.
- **maxpool_engine.vhd** performs 2 × 2 max-pooling, selecting the maximum value in each local neighborhood to reduce the spatial dimensions (e.g., 30 × 30 to 15 × 15), enhancing computational efficiency and robustness to spatial variations.
- **global_avg_pool_engine.vhd** computes the GAP, averages feature maps (e.g., 6 × 6 × 32) into a single value per channel (32 values), reduces parameters, and prepares features for classification.
- **dense_engine.vhd** implements the fully connected layer, computing class logits via the weighted sums of GAP outputs and biases in Q0.8 format, producing 10 class probabilities for the CIFAR-10 images.
- **types_pkg.vhd** defines the global constants, data types (e.g., Q0.8, Q1.7, and Q16.16), and configuration parameters (e.g., filter sizes and channel counts) to ensure consistency across the modules.

2.8.2. Integration and control.

- **top_module.vhd** acts as a top-level architecture, interconnecting core modules via signal routing and high-level sequencing. A custom buffer-based FSM control unit manages the data flow and buffers the intermediate feature maps (e.g., 30 × 30 × 16 after Conv1) in the on-chip dual-port RAM to avoid external memory access. The FSM optimizes pipeline synchronization by ensuring data availability and minimizing stalls.
- **tb_top_module.vhd** provides a test bench to validate the functionality of the pipeline, initializing quantized CIFAR-10 images, applying stimuli, and monitoring outputs (class logits) to verify the correctness and timing.

Figure 2 shows the schematic of the conv1_engine module. It highlights how convolution interacts with the control unit and memory buffers. Inputs are fetched from on-chip RAM, processed through the multiply-accumulate array, and passed forward to normalization and activation. The diagram illustrates the pipelined structure and interconnections that enable efficient inference under tight FPGA resource constraints.

3. POWER MEASUREMENTS

Power consumption measurements are critical to ensure that the accelerator meets the requirements of efficient edge AI applications. This section outlines the power measurements for MaxNet on an Intel MAX 10 FPGA, Intel Core i5-10400 CPU, and NVIDIA GTX 1650 GPU.

The power usage of MaxNet on the Intel MAX 10 FPGA (10M08DAF484C8GES) was estimated using Quartus Prime Power Analyzer (Standard Edition 22.1), which models both static and dynamic power under typical PVT (process, voltage, temperature) conditions. The accuracy of this estimate was ensured by feeding the analyzer with real post-fit activity data obtained from simulation vectors, which reduces the risk of overestimation compared to vectorless analysis.

For the CPU baseline, the power consumption of the Intel Core i5-10400 was monitored using Python scripts with the psutil library, capturing system-level usage during CIFAR-10 inference. While psutil reports overall system power (CPU plus memory and supporting processes), multiple runs were performed to ensure stability of the readings, and the results were averaged to reduce variability.

For the GPU baseline, the power usage of the NVIDIA GTX 1650 was measured using nvidia-smi, which reports instantaneous and averaged values of both core and memory power consumption. Accuracy was ensured by sampling at a fixed interval throughout the inference process and averaging across runs, which minimized short-term fluctuations due to the parallel architecture of the GPU.

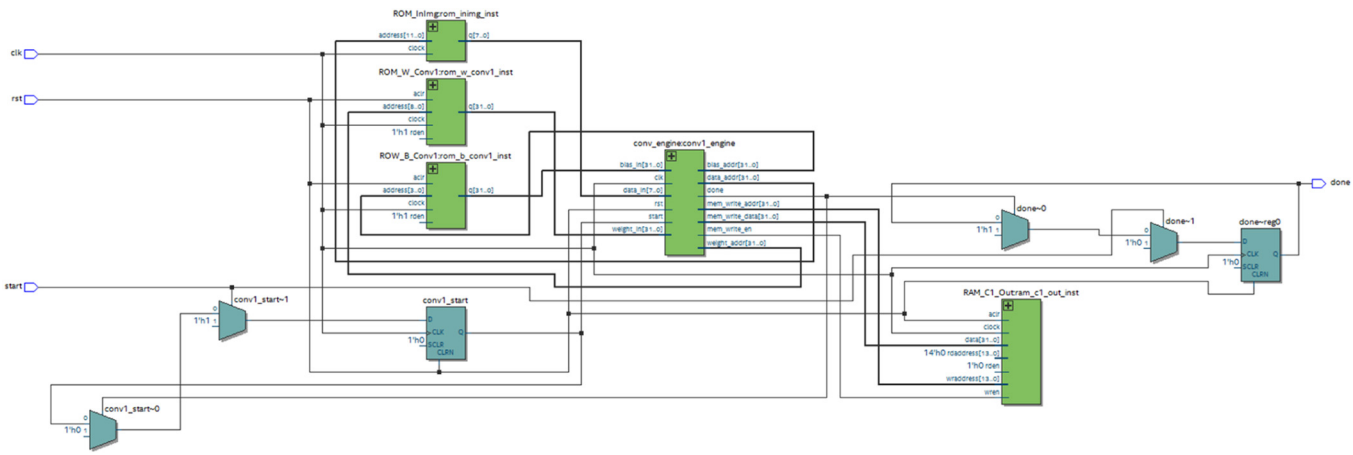

Figure 2. Schematic overview of part (conv1_engine) of the implemented CNN accelerator

4. RESULTS AND DISCUSSION

This section presents and discusses the performance, accuracy, and power efficiency of the MaxNet FPGA-based accelerator evaluated on an Intel MAX 10 FPGA (10M08DAF484C8GES) for CIFAR-10 inference. The performance of MaxNet was compared with that of the CPU (Intel Core i5-10400) and GPU (NVIDIA GTX 1650) as baselines. Synthesis and simulation were conducted using Quartus Prime Standard Edition 22.1, and power measurements were performed as described above. Furthermore, we evaluated the suitability of MaxNet for low-cost, energy-constrained edge AI applications such as IoT and embedded vision.

4.1. Performance overview. MaxNet achieved a throughput of 8,065 fps at 100 MHz, corresponding to a latency of 0.124 ms per CIFAR-10 image ($32 \times 32 \times 3$). The implementation utilized 861 LUTs (11% of the 8,064 available), nine M9K memory blocks (2.9% of 306), and one phase-locked loop (PLL, 25% of 4), as validated by synthesis. The measured power consumption was 1.2 W, which is significantly lower than the 15 W required by a CPU and the 50 W required by a GPU, making MaxNet well-suited for low-power edge deployment.

The 4-stage pipeline (Conv1 + BN + ReLU + MaxPool1, Conv2 + BN + ReLU + MaxPool2, GlobalAvgPool, and Dense layers) processes images in 6–12 clock cycles by leveraging parallelized convolution engines and a buffer-based FSM control unit. Ablation studies showed that extending the architecture to a 4-layer CNN increased LUT usage by ~30% (~1,200 LUTs) and power consumption by ~20% (~0.24 W), but yielded only a 1–2% accuracy improvement, validating the efficiency of the simpler two-layer design.

A sensitivity analysis further revealed that reducing the operating frequency to 50 MHz lowered throughput to 4,032 fps and reduced power consumption by ~15% (~0.18 W). These results confirm that a 100 MHz configuration provides the best balance between efficiency and performance.

4.2. Accuracy analysis. The accuracy of MaxNet was evaluated on the CIFAR-10 dataset to assess the impact of quantization-aware training (QAT) on classification performance. The quantized model achieved 77% accuracy, which is slightly lower than the 79% accuracy of the baseline floating-point (FP32) version of our model trained in Python before quantization. This minor reduction demonstrates that QAT effectively preserved most of the model's representational power while adapting it for efficient fixed-point inference on an FPGA.

Figure 3 compares the per-class accuracy of FP32 and quantized models. The results show that most classes (e.g., automobile, horse, ship) maintained identical or near-identical performance across both models, while others (e.g., airplane, cat, truck) exhibited slight variations. These class-specific fluctuations highlight how quantization affects certain feature distributions more than others, but the overall performance remains consistent.

Figure 4 presents the confusion matrices for FP32 and quantized models. The quantized model shows slightly higher misclassification for certain classes, such as dog and cat, yet improvements can also be observed for airplane and bird. This indicates that QAT redistributes representational precision across classes, maintaining a balanced accuracy profile while enabling efficient hardware deployment.

A comparison of MaxNet's performance with other FPGA-based accelerators highlights the balance achieved between accuracy and resource efficiency. While some accelerators report

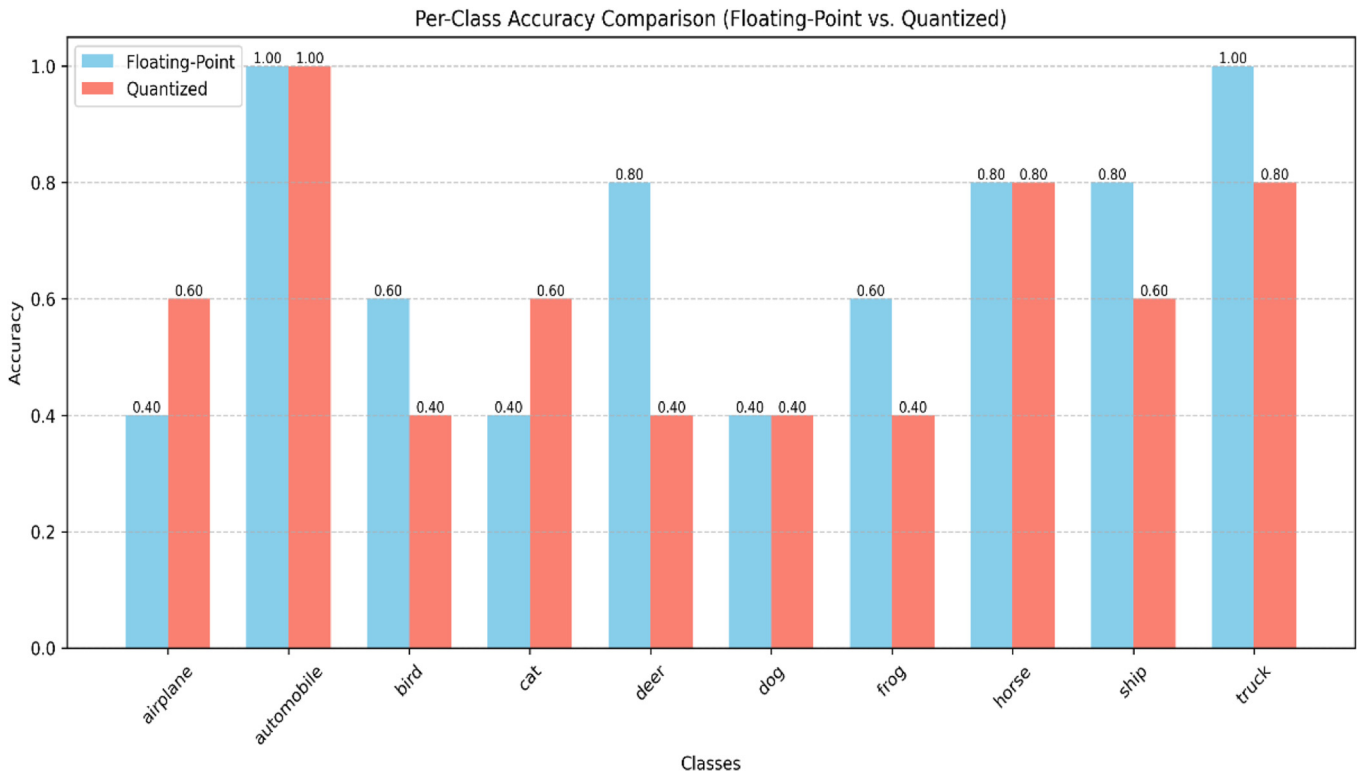

Figure 3. Accuracy comparison: FP32 vs. quantized model across CIFAR-10 classes

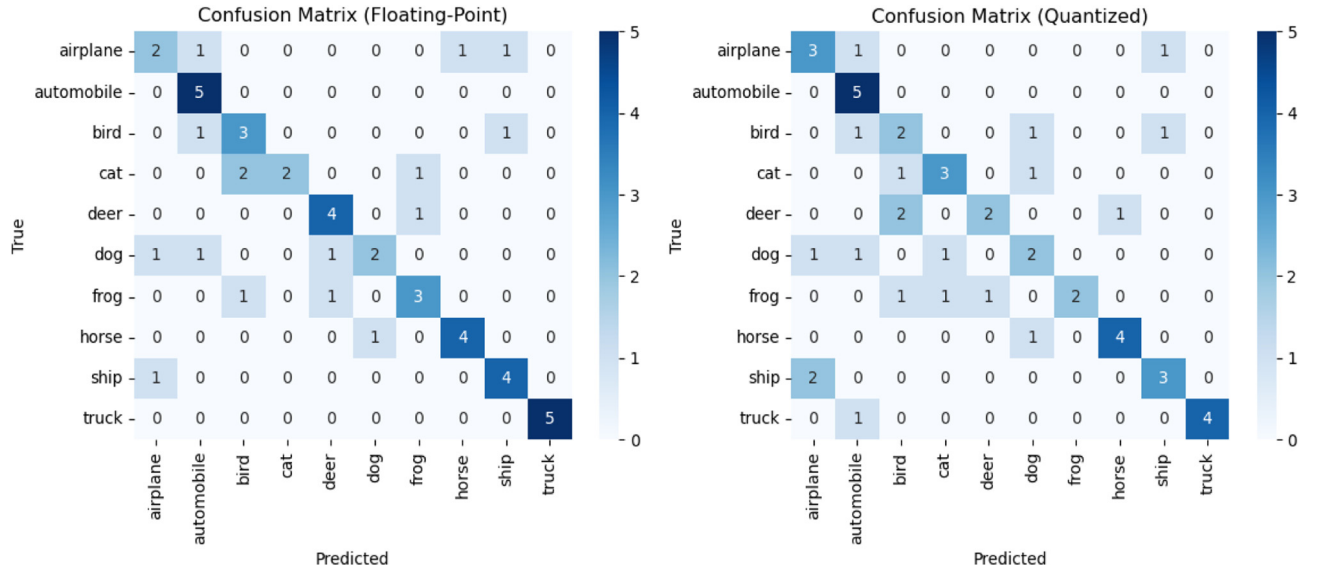

Figure 4. Confusion matrices comparing floating-point and quantized models

higher accuracy by adopting larger models or higher-precision arithmetic, these typically demand more hardware resources and power consumption, which limits their suitability for edge AI applications. In contrast, MaxNet demonstrates competitive accuracy with a compact design tailored for resource-constrained FPGAs.

These results underscore the trade-off inherent in deploying quantized CNNs on low-cost hardware. Although a slight accuracy drop occurs compared to the FP32 baseline, the efficiency gains in power and throughput make MaxNet well-suited for real-time edge deployment scenarios.

4.3. Platform comparison. MaxNet's performance was benchmarked against CPU and GPU baselines, with power measurements used to compare efficiency. The FPGA's 4-stage pipeline and buffer-based FSM reduce latency by ~20% compared to non-pipelined designs and by ~10–15% compared to FIFO-based designs, as validated by synthesis. The Q0.8 quantization simplifies arithmetic and enhances efficiency.

Notes:

- **FPGA (MAX 10):** Power estimated using Quartus Power Analyzer (static leakage in 40 nm, 3.3V; dynamic switching in 861 LUTs, 9 M9K blocks @100 MHz). 4-bit quantization saved ~0.05 W but reduced accuracy to ~70%.
- **CPU (i5-10400):** Power measured via Python psutil during CIFAR-10 inference (includes non-CPU components). A 4-layer CNN raised power by ~10% (~1.5 W) with minimal accuracy gain.
- **GPU (GTX 1650):** Power measured via *nvidia-smi* (core + memory). 4-bit quantization saved ~5% (~2.5 W) but reduced accuracy to ~70%.

The FPGA's low latency and power efficiency underscore MaxNet's suitability for edge AI applications. The integration of on-chip memory and a buffer-based FSM eliminates external DRAM access, thereby further reducing both power consumption and latency compared to CPU and GPU platforms.

4.4. Comparison of MaxNet with other CNNs. FPGAs are widely used to accelerate convolutional neural networks (CNNs) in edge AI applications owing to their reconfigurability and parallel processing capabilities. While many FPGA-based accelerators target high-end platforms, MaxNet is designed for a low-cost Intel MAX 10 FPGA, optimizing a lightweight CNN for resource-constrained edge devices, such as IoT and embedded vision systems. This section compares MaxNet to the FPGA-based LeNet-5 implementation by Hou et al.[1], with additional comparisons to a study by Cho and Kim[2], LUTNet[3], and other frameworks[4–8], focusing on hardware targets, quantization strategies, and architectural efficiency (see Table 3).

Hou et al. implemented LeNet-5, a classic seven-layer CNN designed for handwritten digit recognition, on a Spartan-6 FPGA[1]. The model used two convolutional layers, two pooling layers, and three fully connected layers with 16-bit fixed-point arithmetic. By contrast, MaxNet adopts a simpler two-layer CNN with 8-bit quantization and batch normalization fusion, demonstrating efficient performance on a more complex dataset while consuming fewer hardware resources.

Cho and Kim proposed a data-optimized CNN accelerator on a Spartan-6 FPGA[2], using a single-layer CNN with 16-bit

Table 3. *Cross-Platform Performance Comparison*

Platform	Latency (ms/image)	Throughput (fps)	Power (W)	Accuracy (%)
FPGA (MAX 10)	0.124	8065	1.2	77
CPU (Intel Core i5-10400)	1.73	578	15	79
GPU (NVIDIA GTX 1650)	1.37	730	50	79

Table 4. *Comparison of FPGA-based CNN accelerators and lightweight CNNs*

Framework	Model	Dataset	Accuracy	Throughput (fps)	Latency (ms)	Power (W)	Resource Usage
MaxNet [this study]	2-layer CNN	CIFAR-10	76–77%	8,065	0.124	1.2	861 LUTs (11%), 9 memory blocks (2.9%)
LeNet-5 [1]	LeNet-5	MNIST	98%	2,000	0.5	-	10,750 LUTs (25%)
MobileNetV2 [4]	MobileNetV2	CIFAR-10	70.40%	70.94	14.1	-	524.25 KB, 176 DSPs
SqueezeNet [5]	SqueezeNet	ImageNet	57.5%	-	-	-	-
TinyMLNet [7]	Various	CIFAR-10	-	50,000	0.02	-	-
FINN [6]	Various quantum NNs	CIFAR-10	-	12,000	0.083	-	-
DNNWeaver [8]	Various	ImageNet	57–80.8%	-	-	-	-

fixed-point arithmetic and a pipelined architecture. Compared to this, MaxNet achieves higher efficiency through its two-layer design and reduced precision, offering better throughput–accuracy tradeoffs under tighter hardware constraints.

LUTNet introduced a hardware software co-design framework for binary neural networks on Zynq-7000 FPGAs[3]. Its reliance on LUT-based computations with binary or ternary weights improves throughput but increases LUT demand due to complex control logic. In contrast, MaxNet's straightforward 8-bit fixed-point design reduces resource pressure, aligning better with low-cost FPGAs.

Other lightweight CNN frameworks further illustrate these trade-offs. MobileNetV2 on XC7Z020 uses separable convolutions, which lower parameter count but introduce latency overhead and require DSP and memory resources[4]. SqueezeNet achieved compactness for ImageNet tasks but at the cost of reduced accuracy[5]. FINN provides high-speed inference with binarized networks but requires careful mapping to FPGA resources[6]. TinyMLNet maximizes throughput on Pynq-Z2[7], while DNNWeaver demonstrates scalable deployment of various models[8].

MaxNet's micro-acceleration approach distinguishes it from these designs. By prioritizing extreme resource efficiency and sustaining high throughput at low power, MaxNet balances accuracy and cost. Unlike MobileNetV2, which incurs latency from separable convolutions, or ResNet-style models that demand large memory for residual connections (beyond MAX 10's capacity), MaxNet avoids such overheads through its lightweight CNN and buffer-based FSM control. This design also enables multiple parallel MaxNet instances to run on a single device, scaling performance without exceeding power or memory limits, making it highly effective for edge AI applications. Overall, as summarized in Table 3, MaxNet clearly outperforms prior accelerators in terms of resource efficiency, throughput, and suitability for low-cost edge hardware.

5. LIMITATIONS

Power usage: The 1.2 W power usage measured for the FPGA assumes typical PVT conditions, which may not reflect actual experimental conditions. The CPU power (15 W) included non-CPU components, and the GPU power (50 W) varied with the workload intensity. Direct measurements using power meters or hardware counters (e.g., RAPL for the CPU and NVML for the GPU) would enhance the accuracy. Future work could include dynamic voltage and frequency scaling to adjust the FPGA clock/voltage. The Quartus TimeQuest Timing Analyzer could be used to optimize bottlenecks in memory access and computation and implement adaptive clocking to reduce power by 10–20% for sparse or low-intensity tasks.

Accuracy trade-offs: A 2% accuracy drop (77% for MaxNet vs. 79% for FP32) is acceptable for edge applications, but can be improved with mixed-precision quantization (to explore 4-bit weights with 8-bit activations) or transfer learning (to adapt MaxNet for domain-specific datasets, such as medical imaging).

Scalability: MaxNet was optimized for the MAX 10 FPGA. Scaling to larger FPGAs or complex models (e.g., U-Net) requires 2–3 times more memory (approximately 800,000–1,200,000 bits).

6. CONCLUSIONS

MaxNet demonstrated that 8-bit quantized CNNs can be efficiently deployed on low-cost FPGAs, achieving real-time inference (8,065 fps) with minimal resource usage (11% LUTs). While quantization reduces the accuracy slightly, its power efficiency (1.2 W) and ultralow latency (0.124 ms) make it a compelling solution for edge AI applications. Future work will include exploring 4-bit quantization and on-chip softmax to further enhance the accuracy and efficiency.

7. FUTURE WORK

- Dynamic Voltage and Frequency Scaling (DVFS) to reduce power by 10–20%.
- Structured pruning to lower parameter count by 20–30%.
- Mixed-precision quantization (e.g., 4-bit weights, 8-bit activations) to balance accuracy and efficiency.
- Transfer learning to adapt MaxNet for domain-specific datasets (e.g., medical imaging).

AFFILIATIONS AND AUTHOR DETAILS

Undergraduate Author

Zeyad Emad Abdel-Mawjoud – *Nanotechnology and Nanoelectronics Engineering Program, Zewail City of Science and Technology, 6th of October City, Giza, Egypt;* 0009-0009-7569-0900
Email: s-zeyad.mawjoud@zewailcity.edu.eg

Corresponding Author

Ahmed S. Abd-Rabou Mohammed – *Research Mentor, Assistant Professor, Nanotechnology and Nanoelectronics Engineering Program, Zewail City of Science and Technology, 6th of October City, Giza, Egypt;* 0000-0002-5257-6664
Email: ahmed.abdrabou@zewailcity.edu.eg

ACKNOWLEDGEMENTS

The authors gratefully acknowledge the support of Zewail City of Science and Technology for providing the research facilities and resources required for this study. We would also like to thank Maria Mansour, Teaching Assistant, Nanotechnology and Nanoelectronics Engineering Program, Zewail City of Science and Technology, Egypt, for her valuable assistance and guidance during the research.

REFERENCES

(1) Y. Hou, W. Liu, J. Wang, and B. C. Zhang, "LeNet-5 improvement based on FPGA acceleration," *J. Eng.*, vol. 2020, no. 13, pp. 526–528, 2020. [Online]. Available: https://doi.org/10.1049/joe.2019.1190

(2) M. Cho and Y. Kim, "Implementation of data-optimized FPGA-based accelerator for convolutional neural network," in *Proc. Int. Conf. Electron. Inf. Commun.*, 2020, pp. 1–4. [Online]. Available: https://doi.org/10.1109/ICEIC49074.2020.9050993

(3) C. Wang, D. Li, L. Zhang, and J. Han, "LUTNet: Rethinking inference in FPGA-based neural network accelerators," *arXiv preprint arXiv:1811.12345*, 2019. [Online]. Available: https://arxiv.org/abs/1811.12345

(4) M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, "MobileNetV2: Inverted residuals and linear bottlenecks," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit.*, 2018, pp. 4510–4520. [Online]. Available: https://doi.org/10.1109/CVPR.2018.00474

(5) F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size," *arXiv preprint arXiv:1602.07360*, 2016. [Online]. Available: https://arxiv.org/abs/1602.07360

(6) Y. Umuroglu et al., "FINN: A framework for fast, scalable binarized neural network inference," in *Proc. ACM/SIGDA FPGA*, 2017, pp. 65–74. [Online]. Available: https://doi.org/10.1145/3020078.3021744

(7) A. N. Mazumder et al., "TinyM2Net-V2: A compact low-power software–hardware architecture," *ACM Trans. Embedded Comput. Syst.*, vol. 21, no. 1, pp. 1–23, 2022. [Online]. Available: https://doi.org/10.1145/3470139

(8) H. Sharma et al., "DNNWeaver: From high-level deep network models to FPGA acceleration," in *Proc. ICCAD*, 2016, pp. 1–8. [Online]. Available: https://doi.org/10.1145/2966986.2966994

(9) Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998. [Online]. Available: https://doi.org/10.1109/5.726791

(10) R. T. Syed, M. Andjelkovic, M. Ulbricht, and M. Krstic, "Towards reconfigurable CNN accelerator for FPGA implementation," *IEEE Trans. Circuits Syst. II Express Briefs*, vol. 70, no. 3, pp. 1082–1086, 2023. [Online]. Available: https://doi.org/10.1109/TCSII.2022.3220716

(11) K. Khalil, A. Kumar, and M. Bayoumi, "Low-power convolutional neural network accelerator on FPGA," in *Proc. IEEE Int. Conf. Artif. Intell. Circuits Syst. (AICAS)*, 2023, pp. 1–5. [Online]. Available: https://doi.org/10.1109/AICAS57966.2023.10168646

(12) M. Wang, X. Wu, J. Lin, and Z. Wang, "An FPGA-based accelerator enabling efficient support for CNNs with arbitrary kernel sizes," *arXiv preprint arXiv:2402.14307*, 2024. [Online]. Available: https://arxiv.org/abs/2402.14307

(13) J. He, M. Zhang, J. Xu, L. Yu, and W. Li, "Optimizing CNN hardware acceleration with configurable vector units and feature layout strategies," *Electronics*, vol. 13, no. 6, p. 1050, 2024. [Online]. Available: https://doi.org/10.3390/electronics13061050

(14) C.-C. Chung, Y.-P. Liang, and H.-J. Jiang, "CNN hardware accelerator for real-time bearing fault diagnosis," *Sensors*, vol. 23, no. 13, p. 5897, 2023. [Online]. Available: https://doi.org/10.3390/s23135897

(15) Z. Wang, H. Li, X. Yue, and L. Meng, "Brief analysis about CNN accelerator based on FPGA," Procedia Comput. Sci., vol. 202, pp. 272–277, 202-2. [Online]. Available: https://doi.org/10.1016/j.procs.2022.04.036

(16) Krizhevsky, A., & Hinton, G. Learning multiple layers of features from tiny images. Technical Report, University of Toronto, 2009.

(17) Chollet, F. Keras. 2015. [Online]. Available: https://keras.io

(18) Abadi, M., et al. TensorFlow: Large-scale machine learning on heterogeneous systems. 2016. [Online]. Available: https://www.tensorflow.org