sensors

Communication

FPGA-Based Acceleration on Additive Manufacturing Defects Inspection

Yawen Luo * **and Yuhua Chen**

Department of Electrical and Computer Engineering, University of Houston, Houston, TX 77204, USA; yuhuachen@uh.edu
* Correspondence: yluo12@uh.edu

Abstract: Additive manufacturing (AM) has gained increasing attention over the past years due to its fast prototype, easier modification, and possibility for complex internal texture devices when compared to traditional manufacture processing. However, potential internal defects are occurring during AM processes, and it requires real-time inspections to minimize the costs by either aborting the processing or repairing the defect. In order to perform the defects inspection, first the defects database NEU-DET is used for training. Then, a convolution neural network (CNN) is applied to perform defects classification. For real-time purposes, Field Programmable Gate Arrays (FPGAs) are utilized for acceleration. A binarized neural network (BNN) is proposed to best fit the FPGA bit operations. Finally, for the image labeled with defects, the selective search and non-maximum algorithms are implemented to help locate the coordinates of defects. Experiments show that the BNN model on NEU-DET can achieve 97.9% accuracy in identifying whether the image is defective or defect-free. As for the image classification speed, the FPGA-based BNN module can process one image within 0.5 s. The BNN design is modularized and can be duplicated in parallel to fully utilize logic gates and memory resources in FPGAs. It is clear that the proposed FPGA-based BNN can perform real-time defects inspection with high accuracy and it can easily scale up to larger FPGA implementations.

Keywords: FPGA; additive manufacturing; defects inspection; convolutional neural network; binarized neural network; selective search

check for updates

Citation: Luo, Y.; Chen, Y. FPGA-Based Acceleration on Additive Manufacturing Defects Inspection. *Sensors* **2021**, *21*, 2123. https://doi.org/10.3390/s21062123

Academic Editors: Dadong Wang, Jian-Gang Wang and Ke Xu

Received: 3 March 2021
Accepted: 16 March 2021
Published: 18 March 2021

1. Introduction

Additive manufacturing (AM), involving components production layer by layer, allows users to create a product directly from a 3D computer model, while the conventional subtractive processing is the result of removal of material, formative processes, and joining processes [1–3]. The nature of adding materials from the bottom up makes AM a perfect solution for manufacturing dedicated, complex components. In addition, AM yields faster prototypes and is convenient for a design modification. Moreover, AM makes it possible for object components to come with various colors, textures, and materials. When it comes to manufacturing complex lattice structures, AM is the only solution [4]. Although AM is gaining tremendous attention due to these advantages, it is also more susceptible to internal defects. Therefore, defects inspection is a major concern for AM and it needs to be resolved in real-time automatically.

Defects inspection for traditional subtractive processing is usually performed after the entire object is produced, which typically only guarantees the surface quality. Meanwhile, in some factories, the product inspection is conducted by human vision. It is labor-intensive and the accuracy is limited. Apparently, those methods are not suitable in AM. Chua et al. [5] reviewed current process monitoring/control systems for metal AM and proposed a comprehensive real-time inspection method and a closed-loop monitoring system to improve the quality of AM printed parts. To ensure the internal quality while improving the efficiency of AM product inspection, deep learning is commonly applied in both industry and academia [6–8]. Typically, the deep learning-based defects inspection involves three steps: image acquisition, image classification, and defects localization. Generally, a sophisticated high-speed, high-resolution camera is deployed to capture object surface images during AM processing.

Zhu et al. [9] applied the convolution neural network (CNN) for feature extraction and tried different classifiers: random forest, SVM, and soft-max to achieve 98.75% accuracy on their own dataset. The experimental environment was a 64-bit Windows 10 system with i5 CPU, 8 GB memory, and 2.30 GHz basic frequency. However, their method was time-consuming and could not be applied to real-time industrial applications. Huang et al. [10] introduced a novel compact CNN model with low computation and memory cost and achieved 100% accuracy and 29 ms inference time on the NEU dataset with Intel i3-4010U CPU. Cui et al. [11] presented a CNN model which achieved an accuracy of 92.1% within 8.01 ms. The experiments were conducted with AMD Ryzen 52600 processor and Nvidia 1070 GPU. He et al. [12] proposed a defect detection network (DDN) and set up the NEU-DET dataset. Experiments showed that their DNN obtained 99.67% accuracy for defect classification and 82.3 mAP for defect detection. However, the authors did not present image inference time and hardware configurations. Above researches, while reaching the defects classification goal with high accuracy and promising inference time, were all CPU-based and GPUs were used for acceleration purposes.

Field Programmable Gate Arrays (FPGAs) have high throughput and can inherently provide deterministic low latency for real-time applications [13,14]. Compared to GPUs, FPGAs are more power-efficient and designers can build the neural network from ground up and tailor the FPGA to best fit the model. Moreover, FPGAs allow camera sensor fusion and captured images can be handled by the FPGA directly with minimal latency. ARM processors are generally great for the low-power environment but are typically slower when compared to Intel CPUs [15]. However, modern FPGAs have embedded ARM processors along with custom logic on the same chip, striking a balance between performance and flexibility. In fact, the AM machine itself is an embedded system. It is desirable that the solution can be incorporated as part of the AM machine, so it can perform the defect inspection automatically. This will make the AM machine with defects inspection a standalone system, which is more attractive from the commercial perspective.

Mujawar et al. [16] proposed a 3-layer CNN architecture targeting at written digits recognition application on the MNIST dataset and implemented it in Artix-7 FPGAs. The authors also optimized the architecture by using loop-level parallel processing. Simulation results showed that the FPGA version can achieve 90% accuracy and 58 µs inference time. However, image sizes (28×28) in the MNIST dataset are much smaller than the NEU-DET defects dataset (200×200). Nakahara et al. [17] implemented a multi-scale sliding window-based object detector on an FPGA. They trained the VGG11-based binarized convolutional neural network (BCNN). Their detector could classify the image and locate the object. The implementation was on a Xilinx Zynq UltraScale+ MPSoC zcu102 evaluation board. The researchers did a comparison between their proposed ARM-FPGA detector and the well-known YOLOv2 CPU-GPU version. Their proposed FPGA-based solution could achieve 82.20% accuracy, 28 ms inference time, and consume only 2.5 W. In comparison, the YOLOv2 GPU accelerated version achieved 28.37% accuracy, 20 ms inference time and 250 W.

In the AM process, once any type of defect is identified, AM machine should pause the production immediately and try to further inspect or fix the defect area based on specific defect types. This is because it would be wasting materials and time on a defective product unless the flaws are determined to be acceptable or are repaired. Therefore, locating the defects should be executed right after defects classification. Nakahara et al. [17] used the sliding window on multi-scale images to produce a set of sub-images. After the sub-images were resized to the original size, they were wrapped together and sent to the FPGA-based classifier. For any sub-image labeled with defects, its corresponding coordinates in the original image will depict the defects area. However, the proposed multi-scale sliding window is time-consuming and many defect-free sub-images are generated unnecessarily. In order to produce the potential sub-images with defects more efficiently, region proposal algorithms can be deployed. Van de Sande et al. [18] adapted segmentation as a selective search by reconsidering segmentation and proposed to generate many approximate locations over precise object delineations. Pont-Tuset et al. [19] developed a fast normalized cuts algorithm, and then proposed a high-performance hierarchical segmenter that made effective use of multiscale information. Finally, the authors proposed a grouping strategy that combined multiscale regions into highly-accurate object regions by efficiently exploring their combinatorial space. Taghizadeh et al. [20] proposed an efficient algorithm to generate a limited number of regions for resolving visual recognition problems.

In this paper, we propose an FPGA-based embedded design to perform AM defects identification and classification in real time. Although defects inspection exists, most of the methods were implemented on CPU-GPU and cannot be directly integrated into AM machines. Our method is an embedded FPGA solution and can be readily adopted. The CNN architecture is first presented to achieve high classification accuracy. Next, binarization is applied to CNN parameters to obtain the binarized neural network (BNN) architecture. Then, the BNN is implemented in the FPGA for acceleration purposes. Finally, the FPGA-based BNN model is used for AM defects inspection. Once defects are detected, the selective search algorithm and non-maximum suppression algorithm are applied to locate the defect regions.

The outline of this paper is organized as follows. Section 2 shows research methods. More specifically, defect image acquisition, neural network architecture, FPGA implementation, and defects localization are described. Section 3 presents the results and evaluates the performance. The Section 4 draws the conclusion.

2. Research Methods

2.1. Defect Image Acquisition

Materials used in AM can be polymers and metals [21]. Due to differences in material property, defect features in metal AM and polymer AM have remarkable disparities. In our paper, we studied the metal additive manufacturing. Acquiring the defect image dataset is usually the barrier that stops researchers from training their neural network since obtaining their own dataset from experiments consumes lots of time and materials [22].

2.1.1. NEU-DET

K. Song and his team's efforts on the Northeast University surface defect database (NEU-DET) [23] helped alleviate the above situation in the defects detection field. It includes six kinds of typical surface defects of the hot-rolled steel strip. In our work, we took four types of defects from NEU-DET, namely rolled-in scale, patches, inclusion, and scratches for model training. There are 300 samples for each type of defect, and each image has 200×200 pixels, where the pixel size is 1.975∼2.7 mm (depending on images). The 200×200 image size is typical for training CNNs. Resolutions beyond that, however, will tremendously increase the size of the neural networks and slow down the training and prediction speed. When the image resolution is higher, a common practice is to split the image into sub-blocks and use the sub-images for training the CNN model to perform defect inspection in metal AM [11]. The 200×200 image size used in our paper is comparable to literature with CPU-GPU approaches, and is considered the state-of-the-art comparing to other FPGA-based implementations (e.g., 28×28 in [16], 32×32 and 48×48 in [17]). Zhu et al. [9] applied images with the size of 120×80 pixels to train the CNN for classification of weld surface defects. Our work is applicable to defective images at different scales as long as the images are segmented into sub-images.

2.1.2. Preprocessing

The 200 × 200 image size of the NEU-DET dataset is sufficiently large to train our neural networks. For each image in the NEU-DET dataset, we extracted a defect-free sub-region, resized it to 200 × 200 and labeled it as “defect-free”. In addition, according to the defect annotations, for every image in each defect type, we extracted the sub-regions, resized them into 200 × 200, and labeled them with the same defect type. This ensured that the trained neural network could be also used for locating the defects. The modified dataset was split into training and testing sets at the ratio of 7:3. For every 10 random images in the dataset, seven images were assigned to the training set, and the other three were assigned to the testing set.

2.2. Neural Network Architecture

CNN models are ubiquitous in image data processing. It is one of the state-of-the-art computer vision techniques for image classification [24,25]. In general, the number of parameters in neural networks grows tremendously with the number of layers, which makes the model training computationally intensive. CNNs’ convolutional operations use information from adjacent pixels to effectively downsample the image. It can successfully extract the image features while reducing the number of neural network parameters. Therefore, in this paper, we adopt the CNN model due to its efficiency and high accuracy. However, considering the fact that we will implement it on FPGAs because of the energy efficiency and embedded nature of the design, we binarize the CNN weight parameters. With binarized weight values, we can replace the multiplication with XNOR bit operations, which tremendously speeds up the inference time in the FPGA [26,27]. Therefore, a BNN architecture as shown in Figure 1 is proposed in our studies.

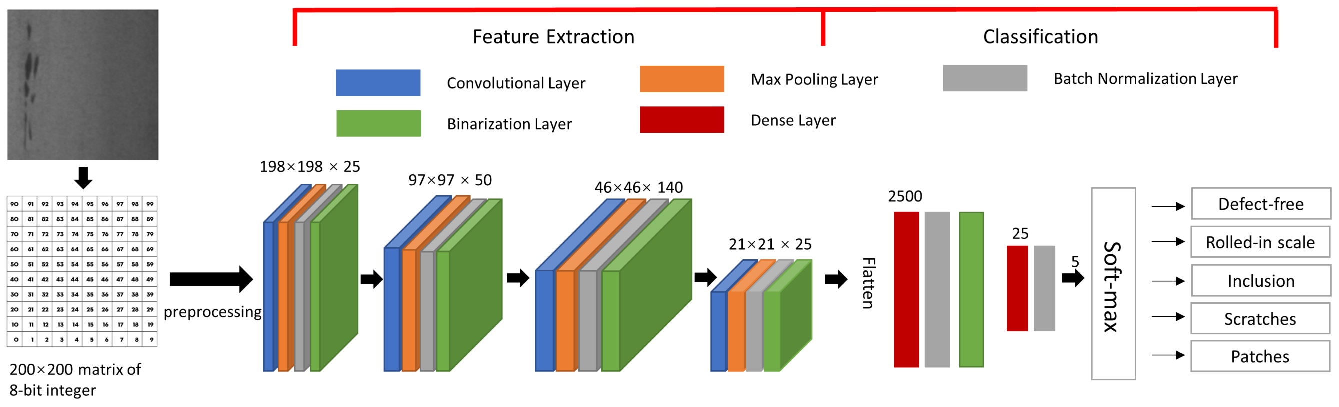


Figure 1. Binarized neural network architecture.

Our BNN model is elaborated as follows. The 200 × 200 pixels gray-scale image is first processed by:

$$f(x) = (x - 127.5)/128. \tag{1}$$

The x is a pixel represented by 8-bit integer, which ranges 0 to 255. The constant 127.5 is the mean of range boundaries 0 and 255. The constant 128 is a scaling factor. After the pre-processing, the pixel values are constrained within $(-1, 1)$. Then the matrices of real numbers are fed into the first convolutional layer with a filter size of 3 × 3 for feature extraction. In the convolutional layer, the “*use_bias*” parameter is set to false. The max pooling layer is used for extracting the maximum value in a sub-region of the feature map. Our max pooling layer is of 2 × 2 in size and the stride is set to 2. The output of max pooling is without padding. Batch normalization layer is applied to mitigate the effect of unstable gradients. It standardizes and normalizes the input values by scaling and shifting operations. In our batch normalization layer, momentum is set as 0.999, epsilon is 0.001, and scale is false, respectively. The momentum is a parameter used for calculating the population average during inference. The epsilon is a small float number added to variance to avoid dividing by zero. The binarization layer works as an activation function as shown in the following equation:

$$q(x) = \begin{cases} -1, & x < 0 \\ 1, & x \geq 0, \end{cases} \tag{2}$$

where x is the output from the batch normalization layer, which is a real number. The $q(x)$ is a 1-bit output of the binarization layer. After passing three additional convolutional layers as well as corresponding max pooling, normalization, and binarization layers, we can successfully extract image features.

Once the features are extracted, the next step is classification. The classification module takes $25 \times 10 \times 10$ feature maps and flattened them as a 2500 feature vector. Then, two dense layers are deployed to densify the 2500 feature vector to the dimensional of 5. Finally, a soft-max layer, as depicted in Equation (3), is applied to generate the output. In our BNN design, M is the number of classes (number of defect types plus one defect-free type). Z_i represents the ith real number from the BNN activation outputs. $\delta(z_i)$ is the probability of the ith class. The final outputs are 5 real numbers representing the probabilities of defect types, namely, defect-free, rolled-in scale, patches, inclusion, and scratches.

$$\delta(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{M} e^{z_j}} \; for\ i{=}1{,}...{,}M\ and\ z = (z_1, ..., z_M) \in R^M. \tag{3}$$

In our BNN model, the cross-entropy loss function, as shown in Equation (4), is applied for measuring the differences between the predicted class and the corresponding labeled class:

$$L(w) = \sum_{m=1}^{M} \sum_{c=1}^{C} y_{mc} log(P(y_m = c|X_m)), \tag{4}$$

where M represents the number of classes, y_m is the predicted class, and c is the labeled class. The images with label m is denoted as X_m. The training process of our model is targeting at minimizing the loss function by adjusting the weight parameters. Our BNN training is performed on an Intel i5-7400 CPU @ 3.000 GHz. We used python scripts with the open-source library of larq [28] and keras [29]. The TensorFlow backend engine was employed, and the NVIDIA GeForce GTX 1060 3GB was applied for acceleration.

2.3. FPGA Implementation

After the BNN model is trained on the CPU, it can be used for inference. In order to make the AM machine an independent and complete system, we present the inference process as an embedded design. The BNN is implemented in the system on a chip (SoC) [30], which is the Terasic SoC kit development board as shown in Figure 2. This board contains an FPGA embedded with an ARM processor, as well as other components such as memory, connectors, sensors, and display. The FPGA device is Intel Cyclone V. The hard processor embedded in the FPGA is a Dual-Core ARM Cortex-A9 MPCore Processor. This board is affordable and is currently available in common electronic components distributors such as Digi-Key Electronics. It can also be ordered directly from Terasic's official website. Due to its low price, it is usually used for education purposes and small system prototypes.

The FPGA acceleration architecture is described in Figure 3. The image prepossessing, convolutional layer 1, and the last activation layer are realized in the ARM. As a result that these layers involve real number arithmetics, it is easier to be implemented in the ARM instead of the FPGA custom logics. The outputs of convolutional layer 1 are sent to the FPGA through the AXI bus using the Xillibus IP core [31]. After FPGA processing, the output of dense layer 2 is then transferred back to the ARM for the activation layer.

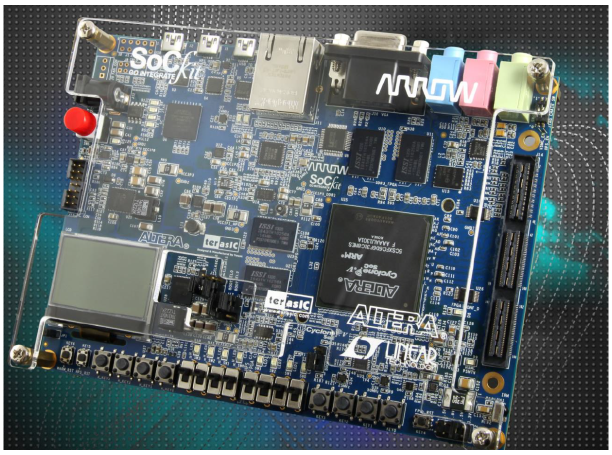


Figure 2. Terasic system on a chip (SoC) kit development board.

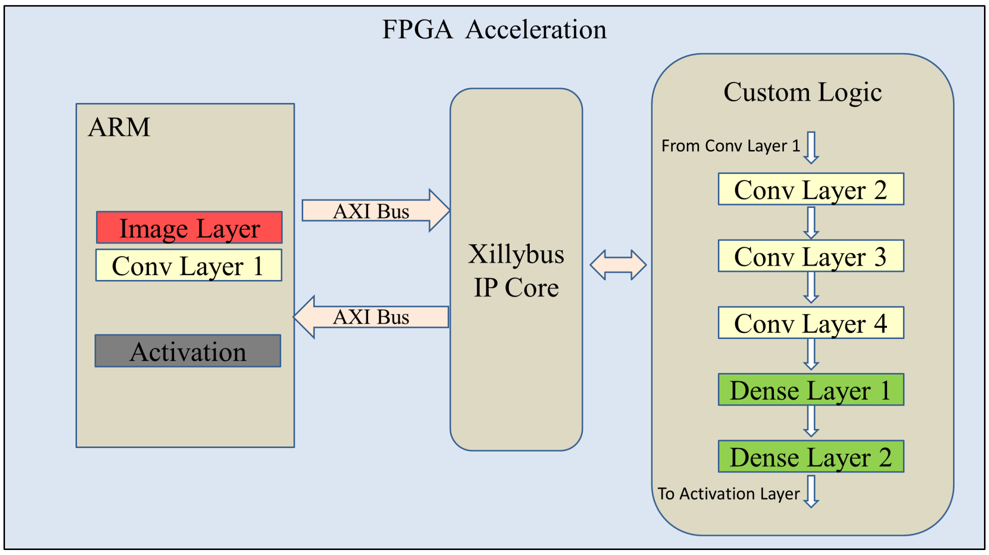


Figure 3. Field Programmable Gate Array (FPGA) acceleration architecture.

In our BNN model, only the convolutional and dense layer weights are binarized, while the parameters of the batch normalization layer are still real numbers. Figure 4 shows the details. Although the drawing only shows part of the entire BNN model, it is clear that the values become real numbers right after passing through the normalization layer, which indicates that the weights of the normalization layer are real numbers. In general, a real number is represented as a floating point number in computers. In order to avoid any floating point operations in the FPGA, which are resource hungry and timing consuming, we use the fixed-point notation [32]. The fixed-point number representation is a real data type for a number that has a fixed number of digits after the radix point. A value of a fixed-point data type is essentially an integer that is scaled by a specific factor determined by the type. In our design, the scaling factor is 1024, which means that 10 bits are needed for representing the original real number's fraction part. According to the maximum value of our batch normalization parameters, we use 18 bits to represent the integer part of the original real number. As a result, the real number is stored as a 28-bit fixed-point number inside the FPGA.

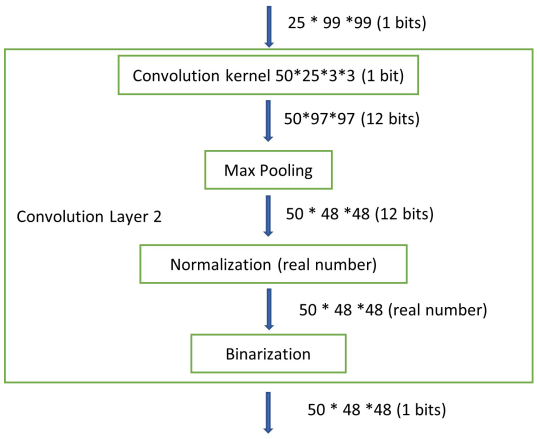


Figure 4. Describes the data type through the binarized neural network (BNN) convolutional layer 2.

Our FPGA design is written in Verilog hardware description language (HDL), which is used to model electronic systems. Verilog is commonly used in the design and verification of digital circuits at the register-transfer level. Different from software design using c/c++, hardware design using HDLs need to take care of the timing. HDLs use principal digital blocks such as flip flops, registers, latches, and memory to describe digital circuits. It has to handle address and clock signals and take the gates/propagation delay into account. In our FPGA design, we use finite state machines (FSMs) to help implement the different convolutional layers in the BNN model. The FSM is a computation model that can guarantee the logic sequence of a function and is usually applied in synchronous designs.

In the FPGA design, the convolution operation can be taken as a set of matrix multiplications. Those intermediate results are stored in the random-access memory (RAM). RAM access time is typically 2 clock cycles. Our convolutional layer filter size is 3 × 3. Before every matrix multiplication, it needs to repeat 9 times the image data access to RAM, which could be time-consuming. Instead, we define a shift register as shown in Figure 5 to access all 9 data at once [17], which speeds up the process dramatically. This will be even more efficient if a larger filter size is deployed (e.g., 5 × 5 or 7 × 7).

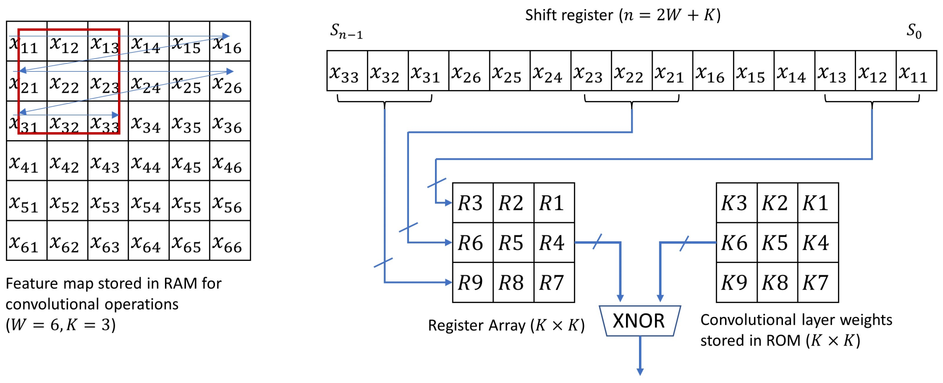

x_{11} x_{12} x_{13} x_{14} x_{15} x_{16}
x_{21} x_{22} x_{23} x_{24} x_{25} x_{26}
x_{31} x_{32} x_{33} x_{34} x_{35} x_{36}
x_{41} x_{42} x_{43} x_{44} x_{45} x_{46}
x_{51} x_{52} x_{53} x_{54} x_{55} x_{56}
x_{61} x_{62} x_{63} x_{64} x_{65} x_{66}
Feature map stored in RAM for convolutional operations ($W = 6, K = 3$)
Shift register ($n = 2W + K$)
S_{n-1}
S_0
x_{33} x_{32} x_{31} x_{26} x_{25} x_{24} x_{23} x_{22} x_{21} x_{16} x_{15} x_{14} x_{13} x_{12} x_{11}
$R3$ $R2$ $R1$
$R6$ $R5$ $R4$
$R9$ $R8$ $R7$
Register Array ($K \times K$)
XNOR
$K3$ $K2$ $K1$
$K6$ $K5$ $K4$
$K9$ $K8$ $K7$
Convolutional layer weights stored in ROM ($K \times K$)

Figure 5. Shift register design.

During the inference process, all the layer weights are stored in the read-only-memory (ROM), which is a type of non-volatile memory used in embedded systems. Data stored in ROM cannot be modified after it is programmed. This makes it perfect to store the trained neural network weights/parameters. In the FPGA design, we need first to prepare the memory initialization files (with the extension .mif) that specify the ROM initial contents.

2.4. Defects Localization

After any image is classified as one of the four defect types: rolled-in scale, patches, scratches, and inclusion, the AM machine should stop printing the object and further inspect the defect areas. To make our AM defect inspection more functionally powerful, we provide the defect locations based on the existing BNN model. The image identified with defects should be retrieved for further processing. Given its size 200 × 200, we use a 4-tuple $< x_{min}, x_{max}, y_{min}, y_{max} >$ that contains four coordinates to constrain the defect area. The x_{min}, x_{max}, y_{min}, and y_{max} are defect area's left boundary, right boundary, up boundary, and bottom boundary, respectively, with the range from 1 to 200. We need to find a 4-tuple value for every defect within that image. Our approach is to suggest as many potential defect sub-regions as possible, where each sub-region is denoted by a 4-tuple. Then those sub-regions will be resized to 200 × 200 pixels and fed into the BNN model. If the output of a sub-region is labeled as defective, its corresponding 4-tuple is marked as one of the defect areas, which can be used for defects repair or displaying on the screen to alert the specialist.

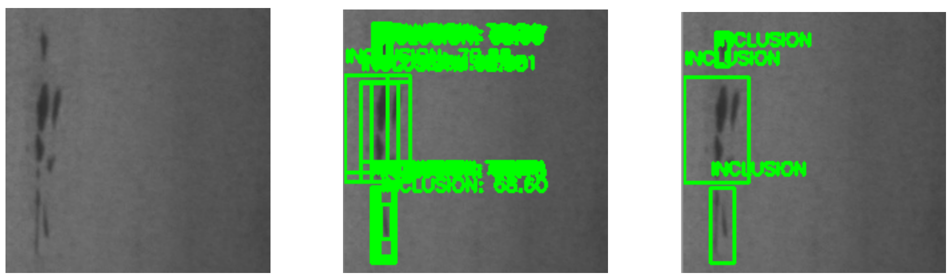
2.4.1. Sub-Region Generation

The objective of sub-region generation is to identify the possible locations of the defects in the image and detect more defects while fewer sub-regions are generated. Selective search algorithm is typically used to efficiently propose multiple sub-regions [19,33]. It starts by over-segmenting the image. Then, it recursively combines smaller similar regions into larger ones. Selective search takes four types of similarities into account, namely, color similarity, texture similarity, size similarity, and shape similarity. When the number of regions, which decreases as the recursive combination continues, reaches a pre-defined threshold, the rest regions are treated as candidate sub-regions.

2.4.2. Non-Maximum Suppression

However, the previously mentioned selective search algorithm will generate many overlapping regions. If one defect object is included within several regions, the BNN would output multiple redundant 4-tuples. More illustrative details are shown in Figure 6. To eliminate those overlapping regions, we adopted the non-maximum suppression [34]. The non-maximum suppression method takes a list of bounding boxes and corresponding defect type probabilities as inputs, where the boxes are described by 4-tuples, and selects a single bounding box out of many overlapping boxes. Non-maximum suppression works as follows:

1. Discard boxes that are below a given probability threshold T_1.
2. Pick the box with the highest probability as the output.
3. Discard remaining boxes where $IoU \geq T_2$ with the previous output box. IoU is the intersection over Union. T_2 is a given threshold, and typically set to 0.5.
4. Repeat step 2 and 3 until no boxes left.



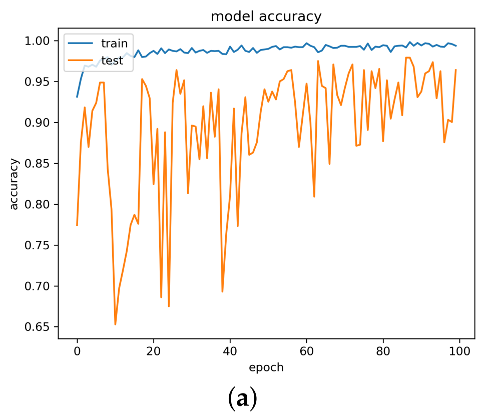
(**a**) (**b**) (**c**)

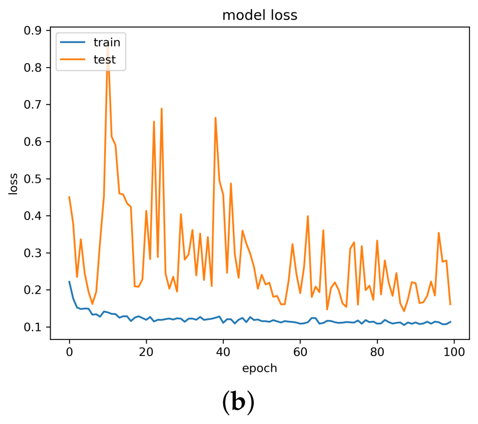
Figure 6. The results of non-maximum suppression algorithm: (**a**) describes the input image; (**b**) describes the defects localization results without non-maximum suppression; and (**c**) describes the defects localization results with non-maximum suppression.

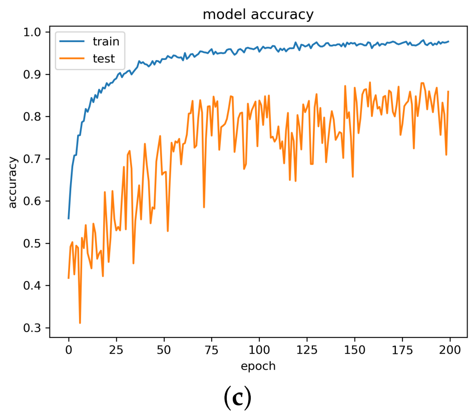
3. Results Evaluation

3.1. Evaluation of BNN Architecture

The testing set images are used for evaluating our BNN model. For every image fed into our BNN model, five outputs are generated to represent the probabilities of each class. The class with the largest probability is our BNN model's prediction. The accuracy is calculated by the number of correct predictions over the total number of predictions. Our BNN model can achieve 97.9% accuracy on either defective or defect-free classification. This is achieved by treating all four defect types: rolled-in scale, inclusion, scratches, and patches as defective only. We also evaluated the classification accuracy on specific defect types for repair purposes. Due to the cross-class similarity, various defect types do not have apparent differences and different types of defects overlap within some images. Even for the same defect class, training defect images are acquired under different illuminations and there are changes in materials. Our defect type classification accuracy is 88.07%. Relative BNN model's accuracy and loss plots are shown in Figure 7. We believe that the 88.07% accuracy on defect type classification is acceptable for our BNN model. In AM defects inspection, the most important goal is to distinguish whether the object has defects or not, in which case the exact defect type is not of high priority.







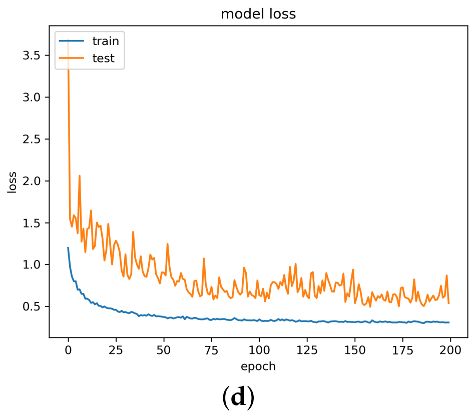
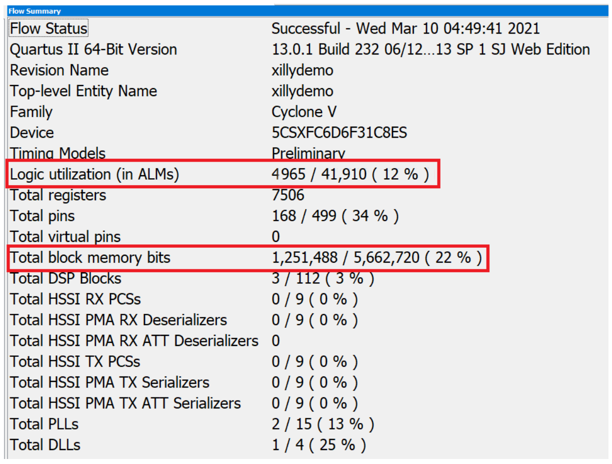


Figure 7. BNN model training results: (**a**,**b**) show BNN model accuracy and loss on defects identification (defective or defect-free); (**c**,**d**) show BNN model accuracy and loss on defect type classification.

3.2. *Evaluation of FPGA Implementation*

Our embedded design of the BNN model is used for inference and a single BNN implementation can inspect one image within 0.5 s. The BNN design is modularized and can be replicated to fully utilize logic gates and memory resources of FPGAs. The FPGA design compilation results are shown in Figure 8. The logic gates utilization is only 12% and block memory utilization is 22%. Each BNN module utilizes 5 K ALMs and 1.25 Mb block memory. Our design can provide real-time inspection capability since AM process is a mechanical process and is relatively slow. The FPGA device on the Terasic SoC kit development board belongs to Intel Cyclone V series, which is a lower-end family of Intel SoC FPGAs. Other SoC FPGA devices (from Intel or Xilinx) can also be used for our BNN model implementation without many modifications. Therefore, our design can easily scale up and inspect more images simultaneously by switching to a more powerful FPGA chip.

For example, Intel FPGA Stratix 10 GX10M can generate 246 BNN modules, which is capable of processing one image every 2 ms. Table 1 shows three Intel FPGA families with different logic gates and block memory sizes.



Flow Summary

Flow Status	Successful - Wed Mar 10 04:49:41 2021
Quartus II 64-Bit Version	13.0.1 Build 232 06/12...13 SP 1 SJ Web Edition
Revision Name	xillydemo
Top-level Entity Name	xillydemo
Family	Cyclone V
Device	5CSXFC6D6F31C8ES
Timing Models	Preliminary
Logic utilization (in ALMs)	4965 / 41,910 (12 %)
Total registers	7506
Total pins	168 / 499 (34 %)
Total virtual pins	0
Total block memory bits	1,251,488 / 5,662,720 (22 %)
Total DSP Blocks	3 / 112 (3 %)
Total HSSI RX PCSs	0 / 9 (0 %)
Total HSSI PMA RX Deserializers	0 / 9 (0 %)
Total HSSI PMA RX ATT Deserializers	0
Total HSSI TX PCSs	0 / 9 (0 %)
Total HSSI PMA TX Serializers	0 / 9 (0 %)
Total HSSI PMA TX ATT Serializers	0 / 9 (0 %)
Total PLLs	2 / 15 (13 %)
Total DLLs	1 / 4 (25 %)

Figure 8. BNN model FPGA design synthesis results.

Table 1. Describes different FPGA chips' capability of replicating BNN modules.

FPGA Device	ALMs	Block Memory	Maximum No. of BNN Modules
Cyclone V 5CSXC6	41.5 K	5.6 Mb	4
Arria 10 GT1150	427.2 K	65.7 Mb	52
Stratix 10 GX10M	3466 K	308 Mb	246

3.3. *Evaluation of Defects Inspection Results*

For every image fed into our BNN inference model, we have five real number outputs. The largest value means the highest probability and its corresponding class is the expected result. For any image identified as defect-free, no further actions are needed. The AM machine just keeps printing the object, and the camera continues to send images to the BNN module. Once any defect is spotted, the AM machine stops printing. Our ARM processor will execute the selective search algorithm to propose multi-regions. After resizing those regions, the ARM will feed them into the BNN and record their corresponding coordinates in the original image. In addition, non-maximum suppression algorithm is adapted to eliminate overlapping areas. Finally, defect areas will be provided and highlighted with boxes. Some sample defect results are shown in Figure 9.

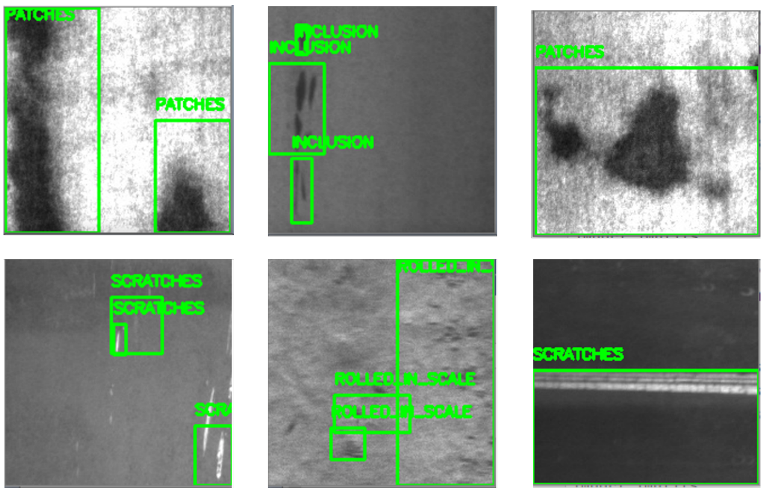


Figure 9. Defect inspection results.

4. Conclusions

In this paper, we have proposed an FPGA-based real-time embedded system for AM defects inspection. The BNN model was presented and implemented on FPGA. Including identifying defect types, our design can also provide defect area coordinates for any follow-up repairs or specialist's re-verification by utilizing selective search and non-maximum suppression algorithms. The NEU-DET dataset is used for training our BNN model. It has achieved 97.9% accuracy in identifying whether captured object image has defects or not. By applying FPGA acceleration, our BNN model can process a single gray-scale image within 0.5 s. Considering the AM object production speed, the results can guarantee real-time quality inspection in the manufacturing process. In addition, our FPGA-based design can easily scale up by generating more BNN modules in a more powerful FPGA. It is also promising for other applications with higher processing speed requirements.

Author Contributions: Conceptualization, Y.L. and Y.C.; methodology, Y.L. and Y.C.; software, Y.L.; validation, Y.L.; formal analysis, Y.L. and Y.C.; resources, Y.C.; data curation, Y.L.; writing—original draft preparation, Y.L.; writing—review and editing, Y.C.; visualization, Y.L.; supervision, Y.C.; project administration, Y.C.; funding acquisition, Y.C. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Acknowledgments: We would like to thank K. Song and Y. Yan for sharing their surface detect database NEU-DET.

Conflicts of Interest: The authors declare no conflict of interest.

Abbreviations

The following abbreviations are used in this manuscript:

AM	Additive manufacturing
CNN	Convolutional neural network
FPGA	Field Programmable Gate Array
BNN	Binarized neural network
NEU-DET	Northeast University Surface Defect Database
HDL	Hardware description language
FSM	Finite state machine
RAM	Random access memory
ROM	Read-only memory
SoC	System on a chip

References

1. Hao, W.; Liu, Y.; Zhou, H.; Chen, H.; Fang, D. Preparation and characterization of 3D printed continuous carbon fiber reinforced thermosetting composites. *Polym. Test.* **2018**, *65*, 29–34. [CrossRef]
2. Wong, K.V.; Hernandez, A. A Review of Additive Manufacturing. *Int. Sch. Res. Notices* **2012**, *2012*, 208760 . [CrossRef]
3. Huang, S.H.; Liu, P.; Mokasdar, A.; Hou, L. Additive manufacturing and its societal impact: A literature review. *Int. J. Adv. Manuf. Technol.* **2013**, *67*, 1191–1203. [CrossRef]
4. Allevi, G.; Capponi, L.; Castellini, P.; Chiariotti, P.; Docchio, F.; Freni, F.; Marsili, R.; Martarelli, M.; Montanini, R.; Pasinetti, S.; et al. Investigating additive manufactured lattice structures: A multi-instrument approach. *IEEE Trans. Instrum. Meas.* **2019**, *69*, 2459–2467. [CrossRef]
5. Chua, Z.Y.; Ahn, I.H.; Moon, S.K. Process monitoring and inspection systems in metal additive manufacturing: Status and applications. *Int. J. Precis. Eng. Manuf. Green Technol.* **2017**, *4*, 235–245. [CrossRef]
6. Chen, Y.; Peng, X.; Kong, L.; Dong, G.; Remani, A.; Leach, R. Defect inspection technologies for additive manufacturing. *Int. J. Extrem. Manuf.* **2021**, *3*, 022002. [CrossRef]

7. Jeyaraj, P.R.; Nadar, E.R.S. Effective textile quality processing and an accurate inspection system using the advanced deep learning technique. *Text. Res. J.* **2020**, *90*, 971–980. [CrossRef]
8. Zhao, L.; Li, F.; Zhang, Y.; Xu, X.; Xiao, H.; Feng, Y. A deep-learning-based 3D defect quantitative inspection system in CC products surface. *Sensors* **2020**, *20*, 980. [CrossRef]
9. Zhu, H.; Ge, W.; Liu, Z. Deep Learning-Based Classification of Weld Surface Defects. *Appl. Sci.* **2019**, *9*, 3312. [CrossRef]
10. Huang, Y.; Qiu, C.; Wang, X.; Wang, S.; Yuan, K. A compact convolutional neural network for surface defect inspection. *Sensors* **2020**, *20*, 1974. [CrossRef]
11. Cui, W.; Zhang, Y.; Zhang, X.; Li, L.; Liou, F. Metal Additive Manufacturing Parts Inspection Using Convolutional NeuralNetwork. *Appl. Sci.* **2020**, *10*, 545. [CrossRef]
12. He, Y.; Song, K.; Meng, Q.; Yan, Y. An end-to-end steel surface defect detection approach via fusing multiple hierarchical features. *IEEE Trans. Instrum. Meas.* **2019**, *69*, 1493–1504. [CrossRef]
13. Kalarot, R.; Morris, J. Comparison of FPGA and GPU implementations of real-time stereo vision. In Proceedings of the 2010 IEEE Computer Society Conference on Computer Vision and Pattern Recognition-Workshops, San Francisco, CA, USA, 13–18 June 2010; IEEE: New York, NY, USA, 2010; pp. 9–15.
14. Psarakis, M.; Dounis, A.; Almabrok, A.; Stavrinidis, S.; Gkekas, G. An FPGA-based accelerated optimization algorithm for real-time applications. *J. Signal Process. Syst.* **2020**, *92*, 1–22. [CrossRef]
15. Ou, Z.; Pang, B.; Deng, Y.; Nurminen, J.K.; Ylä-Jääski, A.; Hui, P. Energy-and cost-efficiency analysis of arm-based clusters. In Proceedings of the 2012 12th IEEE/ACM International Symposium on Cluster, Cloud and Grid Computing (ccgrid 2012), Ottawa, ON, Canada, 13–16 May 2012; IEEE: New York, NY, USA, 2012; pp. 115–123.
16. Mujawar, S.; Kiran, D.; Ramasangu, H. An efficient CNN architecture for image classification on FPGA accelerator. In Proceedings of the 2018 Second International Conference on Advances in Electronics, Computers and Communications (ICAECC), Bangalore, India, 9–10 February 2018; IEEE: New York, NY, USA, 2018; pp. 1–4.
17. Nakahara, H.; Yonekawa, H.; Sato, S. An object detector based on multiscale sliding window search using a fully pipelined binarized CNN on an FPGA. In Proceedings of the 2017 International Conference on Field Programmable Technology (ICFPT), Melbourne, Australia, 11–13 December 2017; IEEE: New York, NY, USA, 2017; pp. 168–175.
18. Van de Sande, K.E.; Uijlings, J.R.; Gevers, T.; Smeulders, A.W. Segmentation as selective search for object recognition. In Proceedings of the 2011 International Conference on Computer Vision, Barcelona, Spain, 6–13 November 2011; IEEE: New York, NY, USA, 2011; pp. 1879–1886.
19. Pont-Tuset, J.; Arbelaez, P.; Barron, J.T.; Marques, F.; Malik, J. Multiscale combinatorial grouping for image segmentation and object proposal generation. *IEEE Trans. Pattern Anal. Mach. Intell.* **2016**, *39*, 128–140. [CrossRef]
20. Taghizadeh, M.; Chalechale, A. Region expansion algorithm: A well-quality region proposal generation. In Proceedings of the 2018 8th International Conference on Computer and Knowledge Engineering (ICCKE), Mashhad, Iran, 25–26 October 2018; IEEE: New York, NY, USA, 2018; pp. 250–255.
21. Marin, F.; de Souza, A.F.; Ahrens, C.H.; de Lacalle, L.N.L. A new hybrid process combining machining and selective laser melting to manufacture an advanced concept of conformal cooling channels for plastic injection molds. *Int. J. Adv. Manuf. Technol.* **2021**, *113*, 1–16. [CrossRef]
22. Bustillo, A.; Urbikain, G.; Perez, J.M.; Pereira, O.M.; de Lacalle, L.N.L. Smart optimization of a friction-drilling process based on boosting ensembles. *J. Manuf. Syst.* **2018**, *48*, 108–121. [CrossRef]
23. Song, K.; Yan, Y. A noise robust method based on completed local binary patterns for hot-rolled steel strip surface defects. *Appl. Surf. Sci.* **2013**, *285*, 858–864. [CrossRef]
24. Wang, J.; Yang, Y.; Mao, J.; Huang, Z.; Huang, C.; Xu, W. Cnn-rnn: A unified framework for multi-label image classification. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Las Vegas, NV, USA, 27–30 June 2016; pp. 2285–2294.
25. Wei, Y.; Xia, W.; Lin, M.; Huang, J.; Ni, B.; Dong, J.; Zhao, Y.; Yan, S. HCP: A flexible CNN framework for multi-label image classification. *IEEE Trans. Pattern Anal. Mach. Intell.* **2015**, *38*, 1901–1907. [CrossRef] [PubMed]
26. Liang, S.; Yin, S.; Liu, L.; Luk, W.; Wei, S. FP-BNN: Binarized neural network on FPGA. *Neurocomputing* **2018**, *275*, 1072–1086. [CrossRef]
27. Kim, J.H.; Lee, J.; Anderson, J.H. FPGA architecture enhancements for efficient BNN implementation. In Proceedings of the 2018 International Conference on Field-Programmable Technology (FPT), Naha, Japan, 10–14 December 2018; IEEE: New York, NY, USA, 2018; pp. 214–221.
28. Geiger, L.; Team, P. Larq: An open-source library for training binarized neural networks. *J. Open Source Softw.* **2020**, *5*, 1746. [CrossRef]
29. Gulli, A.; Pal, S. *Deep Learning with Keras*; Packt Publishing Ltd.: Birmingham, UK, 2017.
30. Pasricha, S.; Dutt, N. *On-Chip Communication Architectures: System on Chip Interconnect*; Morgan Kaufmann: San Francisco, CA, USA, 2010.
31. An FPGA IP Core for Easy DMA over PCIe with Windows and Linux. Available online: Http://xillybus.com/ (accessed on 1 March 2021).

32. Yates, R. Fixed-point arithmetic: An introduction. *Digit. Signal Labs* **2009**, *81*, 198.
33. Uijlings, J.R.; Van De Sande, K.E.; Gevers, T.; Smeulders, A.W. Selective search for object recognition. *Int. J. Comput. Vis.* **2013**, *104*, 154–171. [CrossRef]
34. Hosang, J.; Benenson, R.; Schiele, B. Learning non-maximum suppression. In Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, Honolulu, HI, USA, 21–26 July 2017; pp. 4507–4515.