Journal of Digital Food, Energy & Water Systems [JD-FEWS]

JDFEWS 5 (1): 45-68, 2024
ISSN 2709-4529

Deep Learning Algorithm Analysis of Potato Disease Classification for System on Chip Implementation

John A. ADEBISI
Department of Electrical and Computer Engineering, University of Namibia, Namibia.
adebisi_tunji@yahoo.com

Sesham SRINU
Department of Electrical and Computer Engineering, University of Namibia, Namibia.
ssesham@unam.na

Varqa k. MITONGA
Department of Electrical and Computer Engineering, University of Namibia, Namibia.
varqamitonga@gmail.com

Corresponding Author: adebisi_tunji@yahoo.com

Received: 04 May 2024

Review: 27 May 2024

Accepted: 26 June 2024

Published: 30 June 2024

Abstract - Recently, every aspect of human existence has been affected by modern technologies, including agriculture. The broad range of crops has witnessed setbacks in different capacities due to climate change, leading to diseases and infections and negatively impacting nutrition. Plant diseases significantly threaten agricultural productivity, leading to substantial economic losses and food insecurity. Traditional methods of disease identification, such as visual inspection and laboratory testing, are time-consuming and require expert knowledge. This study aims to address this problem by leveraging Convolutional Neural Networks (CNNs) and FPGA technology for real-time potato disease identification. Five CNN models were trained and evaluated on the Plant Village dataset, achieving accuracies above 90%. ShuffleNet was identified as the most suitable for FPGA deployment, combining high accuracy with low inference time due to its efficient architecture. The selected CNN was implemented on the Xilinx UltraScale+MPSoC ZCU104 board, demonstrating significant power efficiency with a total consumption of 3.333 W, of which 79% was dynamic power. The design met all timing constraints, ensuring reliable operation at a 100 MHz clock frequency, and exhibited low resource utilization with only 0.20% of LUTs, 0.07% of LUTRAMs, 0.13% of flip-flops, and 0.18% of BUFGs used. These results highlight the FPGA's potential as a power-efficient and high-performance alternative to CPU and GPU implementations for real-time CNN inference. Future work includes developing a test bench for detailed performance measurement, optimizing dynamic power usage, and conducting comparative analyses with other platforms. Additionally, expanding the dataset to include more variability and validating the system in real-world applications will further enhance its effectiveness. This research contributes to the goals of Agriculture 4.0 by providing a viable solution for real-time plant disease identification, ultimately aiming at improving agricultural productivity and food security.

Keywords: *Convolutional Neural Networks (CNN), Potato Diseases, High-level Synthesis, System on Chip, Algorithm.*

JDFEWS 5 (1): 45-68, 2024
ISSN 2709-4529

1.0 Background

Potatoes are a globally important crop and contribute significantly to the economies of many countries [1], [2], [3]. In Africa, especially Namibia, potatoes are one of the most consumed horticultural products, accounting for 39% of the total horticultural fresh produce consumption [4]. However, potato diseases such as early and late blight threaten potato production, leading to substantial yield losses (Kanter et al., 2015). Early detection and accurate classification of these diseases are essential for effective disease management and control. Traditional methods, such as visual inspection and expert laboratory testing, are widely utilised. However, such methods are costly and time-consuming [5], [6]. Convolutional Neural Networks (CNNs) have shown promising results in potato disease detection and classification [7]. Thereby, emerging as an effective tool for plant disease classification. CNNs are, however, computationally demanding, which prevents effective real-time detection [8], moreover most of the existing research made significant efforts on software implementation and other platforms such as CPUs and GPUs, which have high power consumption [8], [9], [10]. Field Programmable Gate Arrays (FPGAs) provide a viable solution to these challenges, providing parallel computing and havingow power consumption [9]. Furthermore, System on Chips (SoCs)-which combine the merits of an FPGA and the computational capability of a CPU, have shown efficiency in the implementation of CNNs, compared to other platforms [8].

This study further proposes using an agricultural drone, as illustrated in Figure 1, for precision agriculture to aid images and data collection from crop fields. This framework will be useful for the classification of plant diseases based on the drone images and for producing the required outputs. However, an FPGA could be utilised to accelerate the operations of CNN. The novelty of this work is to investigate the hardware-based analysis of a CNN for early detection and classification of potato diseases and the potential of using FPGAs for real-time detection to prevent the spread of potato diseases. Training and analysis of a CNN algorithm for early detection and classification of potato diseases- specifically early blight and late blight

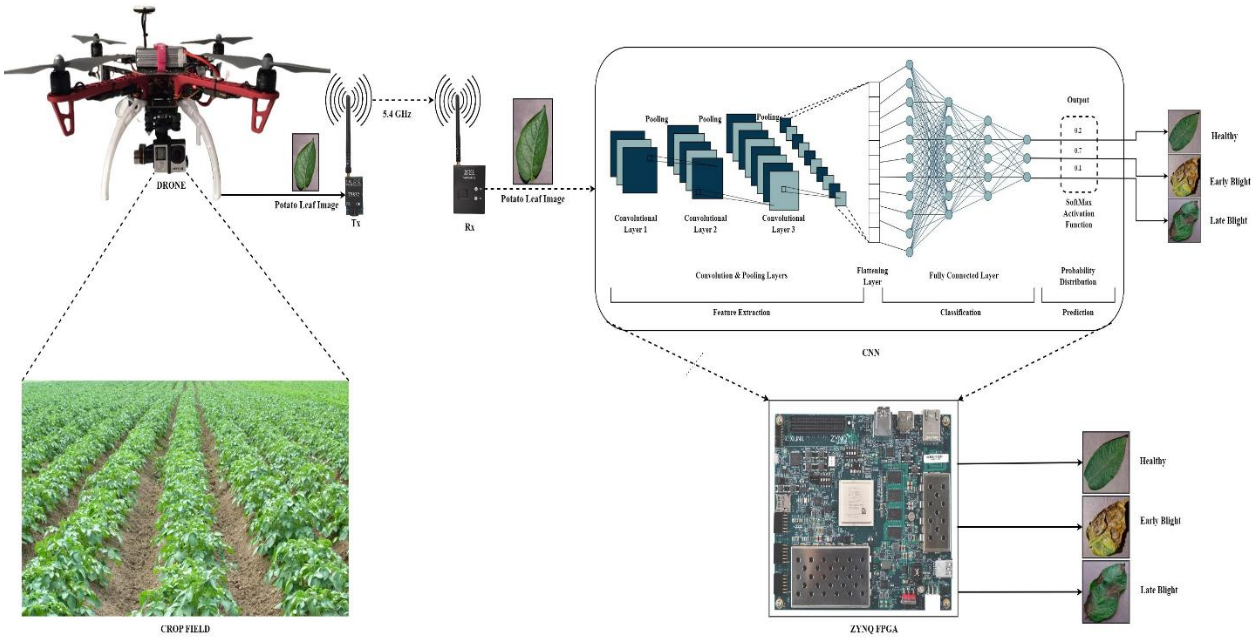


Figure 1: Drone-oriented structure for potato disease detection

2. Review of literature

Related studies regarding the application of CNNs for plant disease detection and its FPGA/SoC implementation were considered in this section. A strategic analysis of the CNN algorithm with respect to layers such as convolutional, pooling, and SoftMax with an overview of metrics utilised to evaluate related performance such as precision, recall, and *F1*-score was considered. As a deep learning algorithm, Convolutional Neural networks have gained widespread adoption in plant disease classification [11]. It has addressed some of the problems associated with traditional classification methods, such as visual inspection or laboratory testing by experts, which are time-consuming and costlier. Other studies have focused on applying CNNs to classify potato diseases [7], [12]. In a study by [13] a CNN algorithm was developed to detect the healthy status of potato leaves. The authors achieved an accuracy of 80%, and their framework was implemented using Python and the Google GPU. Moreover, using MATLAB, [14] compared various deep learning algorithms for potato disease classification. The authors used five classes: healthy, black scurf, common scab, black leg, and pink rot. The detection was based on the surface defects of the potatoes, and the authors reported accurate results. [15] highlighted that CNNs have high computational requirements, and implementation on platforms such as CPUs and GPUs introduce the drawbacks of high-power consumption and low computational efficiency. However, the authors later ascertained that FPGAs are a viable solution to some challenges as they embody the attributes of parallel computing and energy efficiency attributes. Despite these benefits, there is a paucity of hardware implementation of CNN for plant disease detection on FPGA platforms.

[16], implemented a CNN algorithm to identify the diseases of four types of common crops based on their leaves. These crops are beans, rice, coffee, and apple. The author's implementation was carried out on five platforms: FPGA, CPU1, CPU2, CPU2 + GPU and Raspberry Pi. The FPGA-SoC platform was the ZYNQ z7-Lite 7020.In comparison to the other platforms the FPGA recorded the highest accuracy (95.71%), highest speed (0.071s), and lowest power consumption. [17], [18], [19] implemented a CNN algorithm for the identification of three crops: pepper, potato, and tomato. The platform used is PYNQ-Z1 SoC, and GPU boosted CPU was used to train the algorithm. The authors compared the performance of the Zynq-SoC to that of other papers in terms of power consumption, computational roof, computational requirements, and bandwidth roof. The research reported some shortcomings in applying the same to potatoes based on their tested parameters. Most of the reviewed literatures demonstrated the potential for using FPGA-SoCs for real-time potato disease detection, however the scope and focus do not apply to the peculiarity of potatoes based on the findings of this research rather other generality of grains plant diseases. There is paucity in analysing a FPGA-SoC implementation of a CNN for potato disease detection, hence a gap in the literature. The present study aims to fill this gap by proposing a new architectural framework for implementing a CNN algorithm on an FPGA-based SoC to classify early blight and late blight diseases in potato crops.

2.1 CNN Architecture

The architecture of a CNN includes layers which learn the features of image datasets[20], [21], [22]. The convolutional layer is the first layer, carrying out the convolution operation on the input. From this procedure, features from the input data are extracted. The next layer is the pooling layer, which is meant to reduce the size of the resultant matrix from the convolutional layer. It applies either max pooling or average pooling operation. The input then applies non-linearity to the algorithm through an activation function. This allows the network to learn complex patterns in the data.
Furthermore, the bias term is added to the output of the activation function, thereby improving the accuracy of the predictions. The fully connected (FC) layer is the last layer of the algorithm. It takes the output from the preceding layers and gives out a prediction of what the input data. This layer is a type of artificial neural network, which has interconnected neurons. Based on a parameter called the weight, certain neurons are activated based on the features extracted as indicated in Figure 2. While, the SoftMax layer produces a probability distribution, indicating the probability of the prediction.

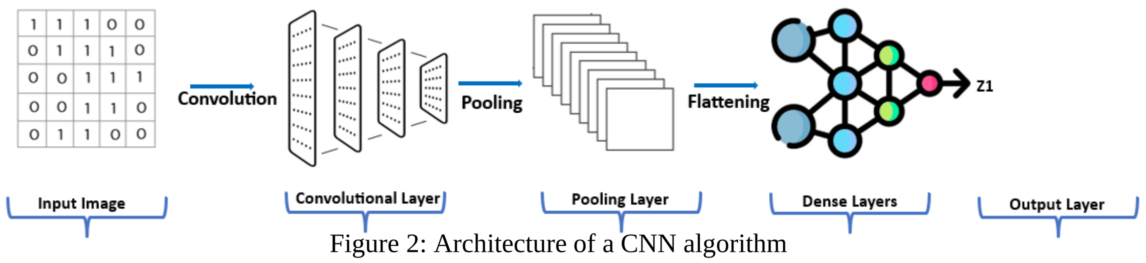


Figure 2: Architecture of a CNN algorithm

It is important to state that in addition to the layers that make up a CNN, there are other building blocks that are essential to the training process [15], [23]. These include forward propagation and backpropagation. The fforward propagation refers to the process of passing an input through the layers of the network to produce an output. The input is passed through the network to produce a prediction during this process. Backpropagation, on the other hand, is the process of computing the gradients of the loss function with respect to the parameters of the network. These gradients are then used to update the parameters of the network, such as the weights and biases in the convolutional and fully connected layers [20], [24].
On the other hand, backpropagation calculates the gradients of the loss or error function of the network. These gradients are utilised to update the parameters of the network to reduce the error in predictions. In convolutional neural networks, the input image is first transformed into a matrix format, which is utilised as input for the network. The transformation corresponds to the type of image representation being used which is the binary notation input illustrated in figure 2. The input model combines 0s and 1s, indicating raw data that can be transformed at various levels. Thus, the input image is converted into a three-dimensional array, whereby the first, second, and third matrix hold different channels. In some implementations, a three-dimensional array could also be fed as input to the network and processed in the relevant layers.

The convolutional layer accepts the input and performs the convolution operation. This layer consists of three types of matrices. The input, the kernel, and the feature map. The kernel, sometimes called the filter or feature detector, is a matrix of smaller dimensions than the input. The values of the kernel are called weights. These weights are used to extract features from the image, such as edges or textures [21], [25]. Convolutional neural networks use both predefined and learned kernels. Predefined or hand-crafted kernels are used to detect edges, textures, and corners. These include vertical and horizontal edge detectors as well as scharr and Sobel filters which is very important when it comes to research of this nature in terms of the potato leaves and its associated features. A vertical edge filter is used to detect vertical edges in the image. This is achieved by searching for abrupt changes in intensity along the vertical direction, which indicates an edge. On the other hand, horizontal edge filters search for abrupt changes in the horizontal direction and are used to detect horizontal edges. The Sobel filter is an edge detection filter, which is based on the gradient of the image intensity. It can detect horizontal and vertical edges, offering more accuracy in detecting diagonal edges. The values and effectiveness of these filters are based on research and so are pre-defined or hand-crafted [25]. Thus, the values of the filters generally do not change during training. In the network, on -predefined filters are also used to detect features from the image rather, they are initialised to certain values. During training, these values will be changed in the convolutional layer until the network begins to effectively and consistently identify an image, thus, the filters are called learned filters. CNNs regularly utilise a combination of both predefined and learned kernels.

2.2 Convolution vs Cross-correlation approach

The convolutional layer takes the input matrix and applies the cross-correlation operation to it. Cross-correlation involves overlapping the kernel on top of the input image and applying the dot product of the two matrices [22], [26], [27]. The kernel is then shifted across the input, repeating the same procedure and producing the feature map. Convolution is a mathematical operation in which the cross-correlation is performed, with the kernel being shifted by 180 degrees. This can be associated with a 7×7 input and a 3×3 filter scenarios. The kernel will be placed on top of the input at the top left corner to perform cross-correlation, and element-wise multiplication will be applied. Then, the kernel will be shifted one step to the right, and the element wise multiplication is performed. Once the filter slides in such a way that its rightmost column is over the right most column of the input, the cross-correlation operation is performed. Then, the kernel shifts one row down and is placed in the left most column of the input. This operation is repeated until the feature map/convoluted feature is obtained. For emphasis, the case of convolution is connected to the kernel rotated by 180 degrees, and then the cross-correlation operation is performed. In addition, multiple filters based on a resultant number of matrices when applied to a cross-correlation input of $n \times n$ with a filter of $f \times f$ dimensions; our output feature map will be of the dimensions:

$$[n - \mathrm{f} + 1\,] \;\times\; [n - \mathrm{f} + 1\,] \qquad (1)$$

Additionally, Padding and strides concepts are applied in CNN to address the exact dimensions scenario with the help of equation (2):

$$p = \frac{f-1}{2} \qquad (2)$$

Where *p* is the padding required, and *f* is the filter size. Stride refers to the step size that shifts the kernel vertically and horizontally across the input image. With padding and strides, an input of $n \times n$ being cross correlated with a filter of $f \times f$, will produce a feature map having the standard dimensions given by equation (3).

$$\left[\frac{\mathrm{n+2p-f}}{\mathrm{s}} + 1\right] \times \left[\frac{\mathrm{n+2p-f}}{\mathrm{s}} + 1\right] \qquad (3)$$

Where *n* is the input size, *p* is the padding, *f* is the filter size, and s is the stride. In the case of RGB images, an input of dimensions: $n \times n \times n_c$ bring cross correlated with a filter of dimensions: $f \times f \times n_c$ will produce a feature map of the following dimensions, where $n_c{}'$ is the number of filters.

$$[n - \mathrm{f} + 1] \times [n - \mathrm{f} + 1] \times n_c{}' \qquad (4)$$

These concepts are all important to achieving target results and can be established using the following mathematical intuition of cross correlation and convolution presented in equations (5) to (12).

Cross-correlation can be represented by the equation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \qquad (5)$$

While convolution takes the equation:

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \qquad (6)$$

Let’s consider the cross-correlation equation:

$$S(i,j) = (I * K)(i,j) = \sum_m \sum_n I(i+m, j+n)K(m,n) \qquad (7)$$

Here:
S is the cross-correlation operation of the functions *K* (kernel matrix) and *I* (input matrix). The double summation is presented because of the 2D signal cross-correlation. And the symbols m and *n* represent the summation indices across the x and y axes, respectively.

Therefore, if we let *m* = *x* and *n* = *y*, the equation can be written as:

$$S(i,j) = (K * I)(i,j) = \sum_x \sum_y I(i+x, j+y)K(x,y) \qquad (8)$$

Considering the input and kernel below:

$$I = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 \\ 1 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 & 1 \\ 1 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 1 \end{bmatrix} \qquad K = \begin{bmatrix} \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \\ \frac{1}{8} & \frac{1}{4} & \frac{1}{8} \\ \frac{1}{16} & \frac{1}{8} & \frac{1}{16} \end{bmatrix} \qquad (9)$$

thus, the restriction of the range of the summations to *x = -1* to *x = 1* and *y = -1* to *y = 1*, if the kernel is a *3 x 3* matrix.

$$S(i,j) = (K * I)(i,j) = \sum_{x=-1}^{1} \sum_{y=-1}^{1} I(i+x, j+y)K(x,y) \qquad (10)$$

Taking the coordinates of the input matrix, 0 is assumed as the center at the index (0,0). Let *I = 0 and j = 0* in the previous equation, this results in equation (11):

$$S(i,j) = \sum_{x=-1}^{1} \sum_{y=-1}^{1} I(x,y)K(x,y) \qquad (11)$$

The above equation is simply the product of the two matrices, as exemplified earlier. Here at *(0,0)* the output is presented in equation (12):

$$S(i,j) = S(0,0) = (K * I)(0,0) = \sum \begin{bmatrix} 1\frac{1}{16} & 0\frac{1}{8} & 1\frac{1}{16} \\ 0\frac{1}{8} & 0\frac{1}{4} & 1\frac{1}{8} \\ 1\frac{1}{16} & 0\frac{1}{8} & 0\frac{1}{16} \end{bmatrix} = \frac{5}{16} \qquad (12)$$

Therefore, to compute the other outputs, the i and j values will be changed to change the position of the kernel. The same element-wise multiplication will be performed. This is the operation which was described in section 2.1. However, the convolution equation (13) can be applied.

$$S(i,j) = (K * I)(i,j) = \sum_m \sum_n I(i-m, j-n)K(m,n) \qquad (13)$$

The only difference is the negative notation. Therefore, the operation will involve the kernel being rotated 180 degrees, and the cross-correlation operation will be performed as discussed earlier. The pooling layer is used to reduce the size of the matrices, accelerate the computations, and make feature detection more robust. There are two types of pooling, maximum pooling, and average pooling [28], although max pooling is mostly utilised. Maximum pooling returns the maximum value from a section of the image covered by the filter while average pooling returns the average of the values from that section. The convolutional and pooling layers introduces invariance to the input image. This can be translational, rotational, or matrix scaling. Therefore, a CNN can have the ability to classify an image even if it has been transformed. So, there will be no need to train the network again for classification involving a transformed image.

2.3 Non-Linear Activation Functions and Bias

To effectively classify the potato leaves, it is important to emphasize the aspect of non-linear activation functions as a critical component of convolutional neural networks (CNNs). Real-world data, often have non-linear relationships between input features and output classes. Since this research considers drone use to capture images as a form of implementing the architecture in Figure 1 in the context of image classification, the relationship between the pixel values of an image and its class is highly non-linear. This is because the values of the pixels are not always the same for a given pixel to correspond to a particular class [29], [30], [31]. Non-linearity enables the classifier to classify the image accurately despite these variations in pixel values.

Gradient descent with momentum is an optimization algorithm commonly used to train neural networks, and some of the features are considered during the data analysis of this research approach. It is an extension of the standard gradient descent algorithm that adds a momentum term to the parameter update rule. The momentum algorithm contains a quantity known as accumulation and is denoted by v_t.This is obtained using the gradient of the current loss, the learning rate, and the preceding value of the accumulation as observed below:

$$v_t = \alpha v_{t-1} - \eta \nabla J(\theta) \qquad (14)$$

The preceding value of accumulation,v_{t-1} is scaled by a constant α, which is referred to as momentum. The momentum is set to a value between 0 and 1 and represents the level at which the preceding step influences the current step. It is usually set to a value of 0.9. Once the accumulation is calculated, the parameters are updated using equation (15).

$$\theta_t = \theta_{t-1} + v_t \qquad (15)$$

Accumulation affects the rate of convergence in that if the minimum is approached quickly, then αv_{t-1} will be significant, and the minimum will be reached even faster. If the optimizer rests between two values, then αv_{t-1} will minimize the number of times the parameters are updated. Further simplification of equation (15) led to the Nesterov Accelerated Gradient descent algorithm (NAG), which alters the momentum algorithm by updating the parameters before the loss is calculated as given by equation (16), which has a higher convergence rate compared to the gradient descent algorithm.

$$\begin{aligned} v_t &= \alpha v_{t-1} - \nabla J(\theta - \alpha v_{t-1}) \\ \theta_t &= \theta_{t-1} + v_j \end{aligned} \tag{16}$$

The same learning rate is applied to each parameter being trained in both the gradient descent and momentum algorithms. However, different parameters can converge to their minimum at different learning rates. The adaptive gradient (Adagrad) algorithm considers learning rate alters from parameter to parameter and step to step. The tth step for the ith parameter is represented as $n_{t,i}$ while the algorithm calculates subgradients instead of gradients. This is a generalization of a gradient that is applicable to nondifferentiable functions. Therefore, Adagrad can be used to optimize both differentiable and nondifferentiable functions, as presented in this research. [32], [33] developed the first Adagrad algorithm. The equation for the learning rate of a given parameter is given by equation (17).

$$\eta_{t,i} = \frac{\eta}{\sqrt{G_{t,ii}}} \tag{17}$$

In equation (17), $G_{t,ii}$ is the ith element of the diagonal of a matrix that is made by computing the product of the subgradient of the loss with itself. After the learning rates are calculated, the algorithm then updates the parameters as $\theta_{t,i} = \theta_{t-1,i} - \eta_{t,i} g_t$. Although learning rates always decrease in size as the training progresses, their values will reach zero at a point, which will stop the training and sometimes prompt the use of combinational techniques. Considering the various reviews of approaches and concepts in CNN.

3.0 Methodology

The samples used in this research consist of potato plant images with early blight and late blight diseases from the PlantVillage dataset, which serves as the study population. For the training and testing process, a random sampling procedure is used to divide the dataset into two separate sets: a training set and a testing set. This is done by randomly selecting a proportion of images for the training and testing set. The random sampling procedure is used due to the accuracy of each image in the dataset, which has an equal chance of being included in either the training or testing set. This helps to minimize any potential bias in the data and ensures that the resulting model is more representative of the population, as indicated in section 2. MATLAB is used to train and analyze the CNN algorithm on the Plantvillage dataset. Furthermore, Vitis High-level Synthesis (HLS) converts the CNN model from its MATLAB form into Hardware Descriptive Language (HDL). The methodology is summarised in Figure 3.

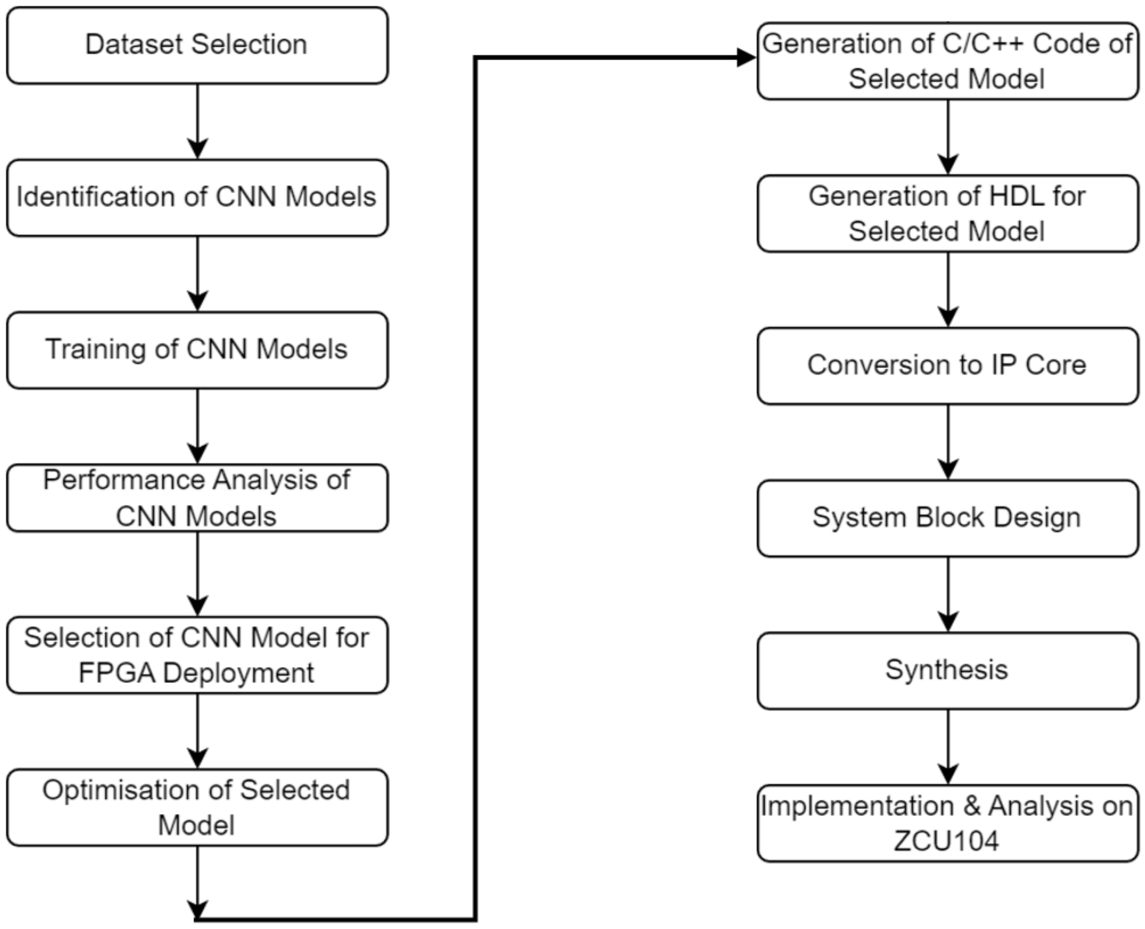

Dataset Selection
Identification of CNN Models
Training of CNN Models
Performance Analysis of CNN Models
Selection of CNN Model for FPGA Deployment
Optimisation of Selected Model
Generation of C/C++ Code of Selected Model
Generation of HDL for Selected Model
Conversion to IP Core
System Block Design
Synthesis
Implementation & Analysis on ZCU104

Figure 3: Research Methodology

3.1 Data collection and analysis

This research considered Five CNN models for training on the Potato Disease dataset, each representing distinct architectural configurations and hyperparameter settings. The training was conducted in MATLAB, leveraging its computational capabilities for machine learning tasks. Subsequently, a comparative analysis was performed to assess the models' performance using key performance metrics, including accuracy, precision, and recall. This comparison aimed to identify the most effective model for further deployment. The chosen model was then prepared for deployment on the Zynq UltraScale FPGA-SoC. To meet the hardware requirements, the model was converted to HDL with a detailed performance analysis of the synthesized model on the FPGA platform. This analysis, utilizing various performance metrics, aimed to evaluate the efficiency and feasibility of deploying the CNN algorithm on FPGA hardware. The stepwise process, an encompassing model training, selection, conversion to HDL, and FPGA synthesis analysis forms a comprehensive approach to implement and evaluate a CNN algorithm for potato disease classification on FPGA-based SoC. Performance metrics played a crucial role in assessing and comparing the models at different stages of this process. PlantVillage is a publicly accessible dataset that contains over 20,639 images of images of crop leaves, including diseased ones. Each image is labeled as either healthy or diseased (figure 4).

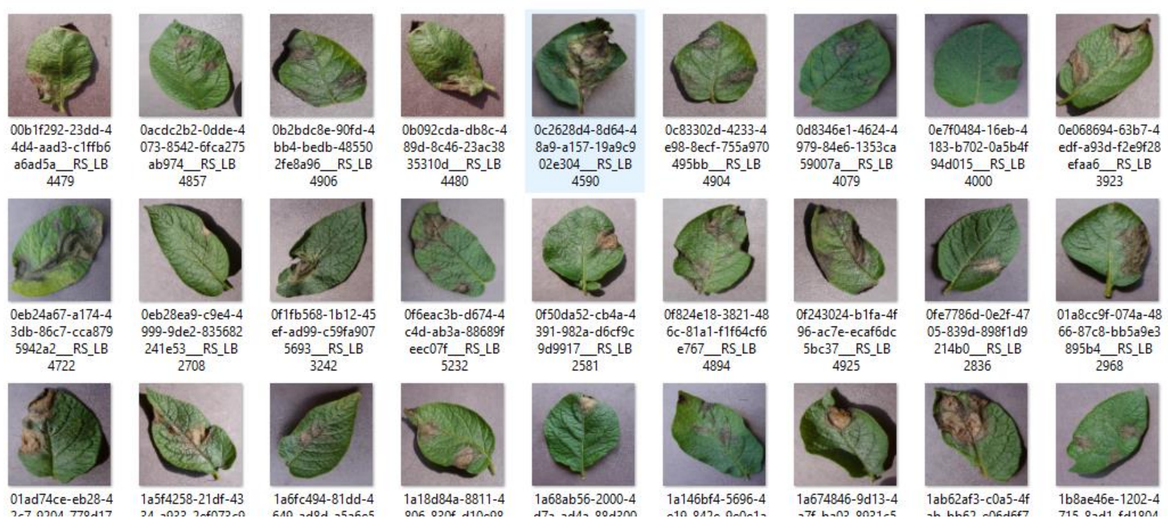

00b1f292-23dd-44d4-aad3-c1ffb6a6ad5a___RS_LB 4479
0acdc2b2-0dde-4073-8542-6fca275ab974___RS_LB 4857
0b2bdc8e-90fd-4bb4-bedb-485502fe8a96___RS_LB 4906
0b092cda-db8c-489d-8c46-23ac3835310d___RS_LB 4480
0c2628d4-8d64-48a9-a157-19a9c902e304___RS_LB 4590
0c83302d-4233-4e98-8ecf-755a970495bb___RS_LB 4904
0d8346e1-4624-4979-84e6-1353ca59007a___RS_LB 4079
0e7f0484-16eb-4183-b702-0a5b4f94d015___RS_LB 4000
0e068694-63b7-4edf-a93d-f2e9f28efaa6___RS_LB 3923
0eb24a67-a174-43db-86c7-cca8795942a2___RS_LB 4722
0eb28ea9-c9e4-4999-9de2-835682241e53___RS_LB 2708
0f1fb568-1b12-45ef-ad99-c59fa9075693___RS_LB 3242
0f6eac3b-d674-4c4d-ab3a-88689feec07f___RS_LB 5232
0f50da52-cb4a-4391-982a-d6cf9c9d9917___RS_LB 2581
0f824e18-3821-486c-81a1-f1f64cf6e767___RS_LB 4894
0f243024-b1fa-4f96-ac7e-ecaf6dc5bc37___RS_LB 4925
0fe7786d-0e2f-4705-839d-898f1d9214b0___RS_LB 2836
01a8cc9f-074a-4866-87c8-bb5a9e3895b4___RS_LB 2968
01ad74ce-eb28-4
1a5f4258-21df-43
1a6fc494-81dd-4
1a18d84a-8811-4
1a68ab56-2000-4
1a146bf4-5696-4
1a674846-9d13-4
1ab62af3-c0a5-4f
1b8ae46e-1202-4

Figure 4: Potato Leaves Dataset

There are 15 directories representing classes of the images (figure 4), including those of pepper, potatoes, and tomatoes. In the case of the potatoes, the dataset includes three directories tallying to 2,150 images, of which 1,000 images belong to the early blight and late blight categories, respectively while the remaining 150 images belong to the healthy class. The dataset is the most widely used dataset by researchers for crop disease classification applications. This is because it is publicly accessible-unlike many others- and the directories are large, making them robust and effective for the task. Hence, the Plantvillage dataset for potatoes was selected for this research. Five lightweight CNN models were trained to evaluate and select the most suitable model for potato disease classification and FPGA implementation. The chosen models were AlexNet, SqueezeNet, ShuffleNet, MobileNet and ResNet18. For the training and validation of each model, 80 % of the Plantvillage dataset was used for training, and 20% was used for validation. Furthermore, data augmentation was used to prevent overfitting, which included rotations, translations, shear, and scale. A diagrammatic comparison of the original and augmented images is presented in Figure 5.

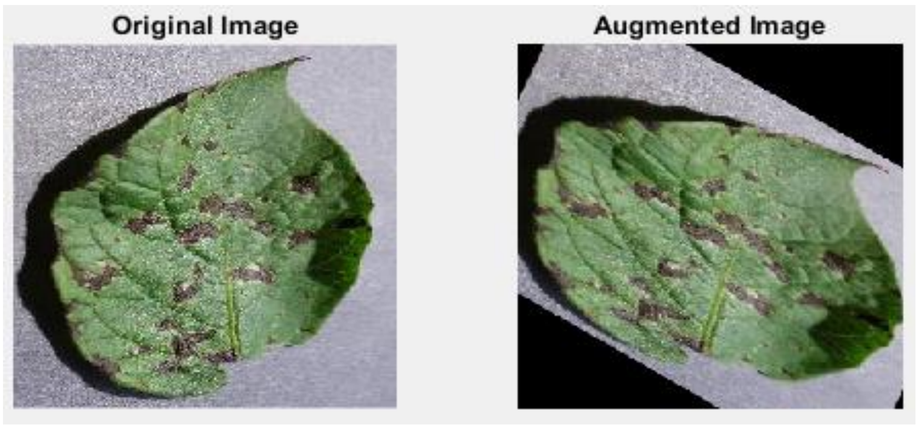
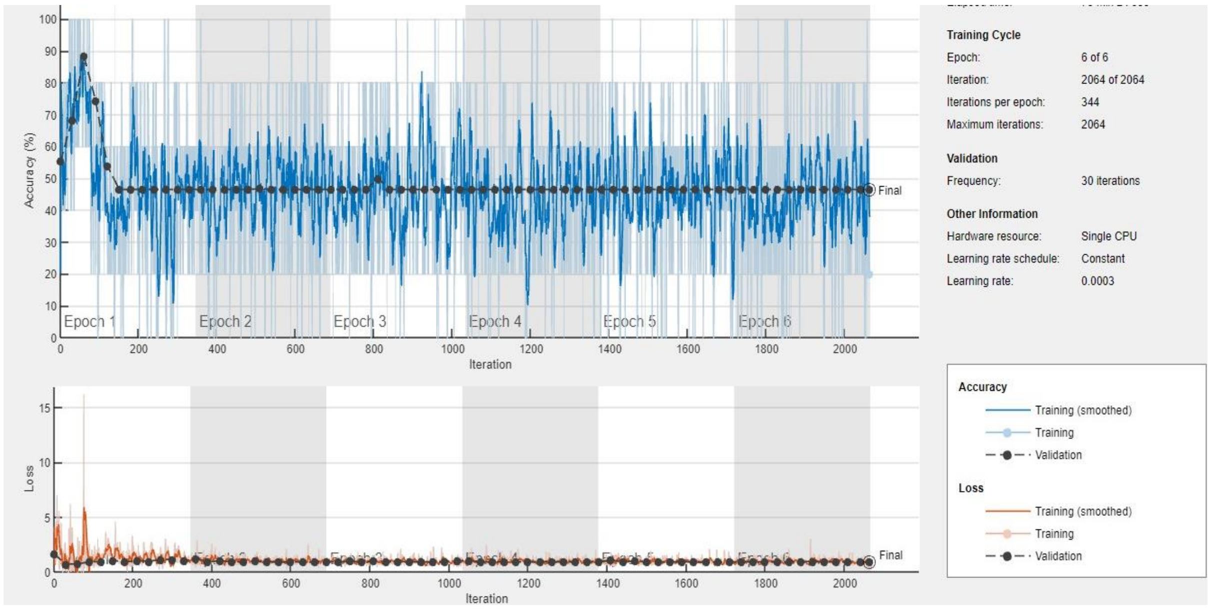


Figure 5: Original vs Augmented Image

The models were trained using the following training parameters: Learning rate: 0.0001, Batch size:5, Epochs:6, and the Adam optimizer. Initially, the models had low validation accuracy, indicating overfitting or a high learning rate. Furthermore, data augmentation techniques were used to prevent overfitting.

4.0 Discussion of Results

Following the successful acquisition of the dataset. The training was administered as indicated in Figures 6 – 8. This includes the training times and the five different CNN models tested. During the training sessions, 6 epochs were completed successfully with 344 iterations of epoch. Training results indicated that the losses were kept at the barest minimum as shown in figures 5 and 6. Results presented also affirm the literature facts that the better the performance of the hardware, the faster the results are generated during the training session. The plot of time taken vs. CNN models presented in Figure 8 indicated the minimum time taken by squeeze net is 33 minutes, while other models took longer to the tune of even over an hour. This result is based on the specification of the hardware used for training the model. The results of the MATLAB simulation vice a vis the FPGA indicate a positive disease detection with an accuracy of over 90%. Details of the simulation are presented in Figures 9-16.



Training Cycle

Epoch:	6 of 6
Iteration:	2064 of 2064
Iterations per epoch:	344
Maximum iterations:	2064

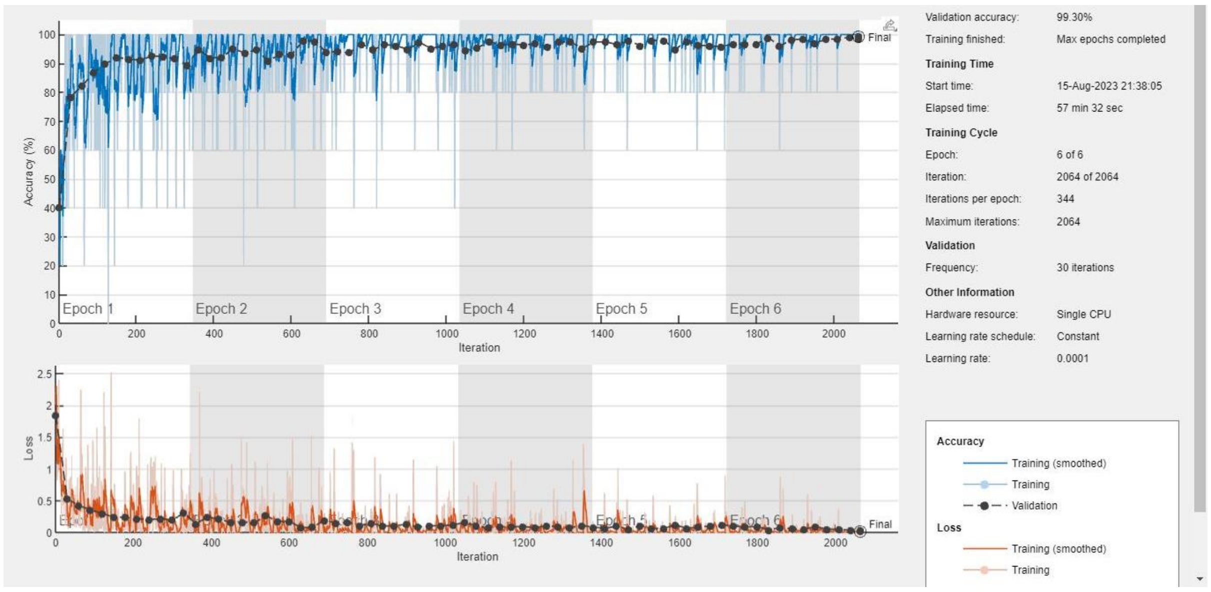
Validation

Frequency:	30 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0003

Figure 6: MobileNet training initial training interface



Validation accuracy:	99.30%
Training finished:	Max epochs completed

Training Time

Start time:	15-Aug-2023 21:38:05
Elapsed time:	57 min 32 sec

Training Cycle

Epoch:	6 of 6
Iteration:	2064 of 2064
Iterations per epoch:	344
Maximum iterations:	2064

Validation

Frequency:	30 iterations

Other Information

Hardware resource:	Single CPU
Learning rate schedule:	Constant
Learning rate:	0.0001

Figure 7: MobileNet training progress after adjustments

Figure 8 shows the training times of the five respective CNN models. Since the CNN model was trained on a core *i5* CPU + 3GB GPU, the models took on average, more than an hour to train. These findings support the literature stating that CPU platforms are inefficient for CNNs.

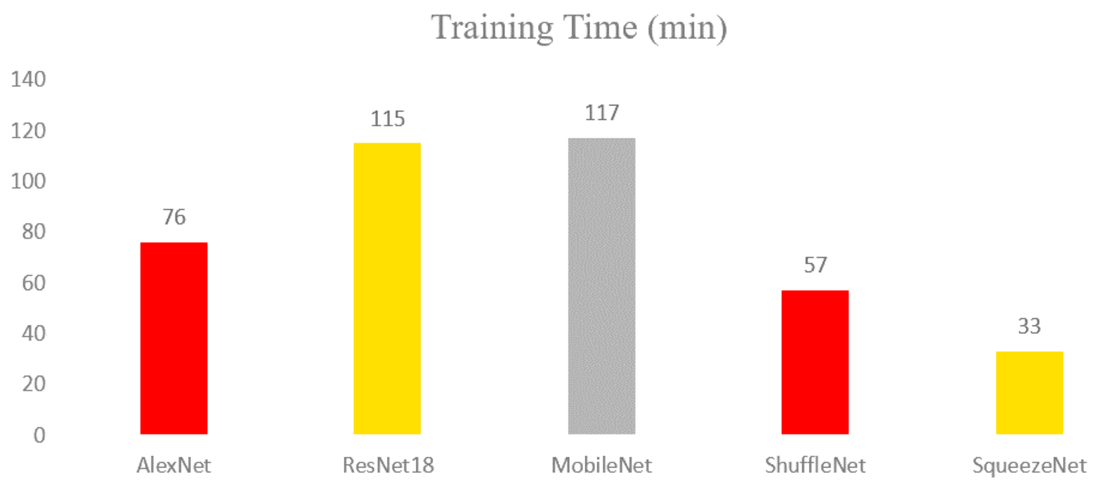


Figure 8: Training times of the CNN models

Following successful training, the models were validated and evaluated using multiple performance metrics as well as ShuffleNet computational requirements (Table 1 and Table 2), including precision, F1-score, recall sensitivity, and specificity. Moreover, the hardware requirements were evaluated using metrics such as number MACs, arithmetic intensity, and FLOPs. Figure 9 shows various statuses and times in a confusion matrix illustration. The hardware requirement that produces this result and its associated performance metrics are shown in Figure 9.

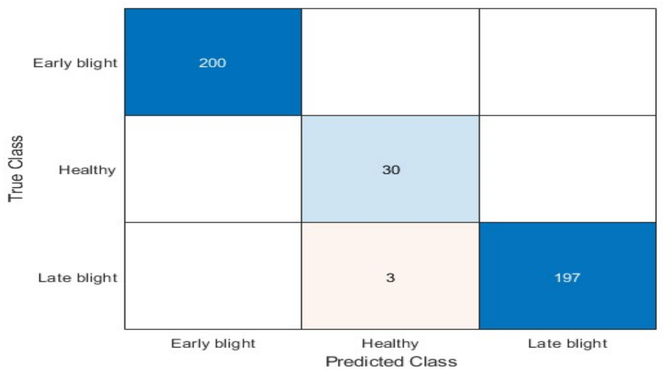


Figure 9: ShuffleNet Confusion matrix

Table 1: ShuffleNet performance metrics

	Precision	Recall	F1-Score	Sensitivity	Specificity
Early Blight	1	1	1	1	1
Healthy	0.90909	1	0.95238	1	0.9925
Late Blight	1	0.985	0.99244	0.985	1

Table 2: ShuffleNet computational requirements

Model	Number Of Learnables	Number Of Operations	Parameter Memory (MB)	Number Of MACs	Arithmetic Intensity	FLOPs	Inference Time (ms)
ShuffleNet	836515	248244320	3.1910515	124122160	870.63658	248244320	0.99297728

It can be deduced that ShuffleNet achieves a very high performance since all the performance metrics (i.e. precision to specificity) are all close to 1. However, considering similar computational requirements, the sensitivity of healthy and late blight diseases in the potato is very close, indicating accuracy in prediction (Figure 10 and Tables 3 and 4).

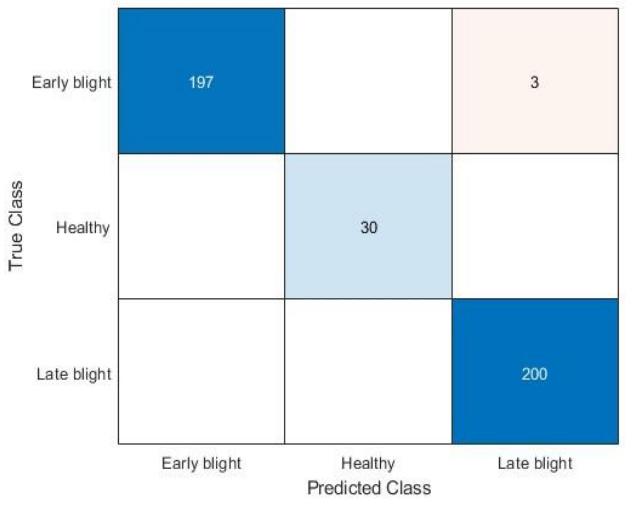


Figure 10: ShuffleNet Confusion matrix

Table 3: MobileNet performance metrics

	Precision	Recall	F1-Score	Sensitivity	Specificity
Early Blight	1	0.985	0.99244	0.985	1
Healthy	1	1	1	1	1
Late Blight	0.98522	1	0.99256	1	0.98696

Table 4: MobileNet computational requirements

Model	Number Of Learnables	Number Of Operations	Parameter Memory (MB)	Number Of MACs	Arithmetic Intensity	FLOPs	Inference Time (ms)
MobileNet	2210659	598996224	8.4329948	299498112	1343.2517	598996224	2.395984896

MobileNet, similarly, achieves high accuracy as suggested by all the metrics being close to 1. While figure 11 and tables 5 and 6 depict the corresponding performance matrix.

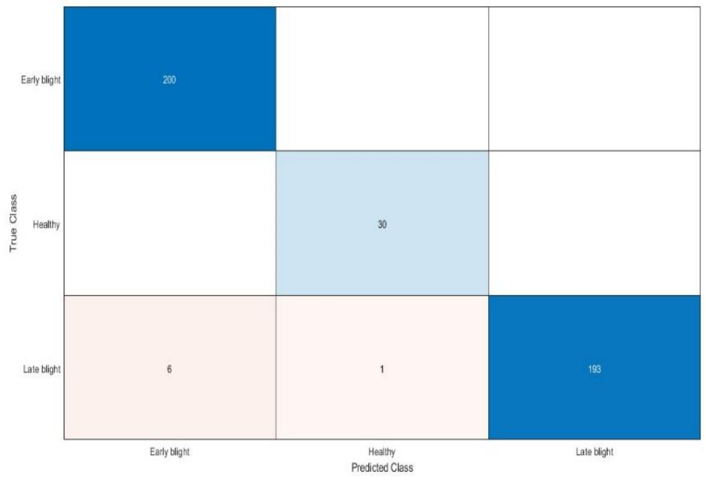


Figure 11 : ResNet18 Confusion matrix

Table 5: ResNet18 performance metrics

	Precision	Recall	F1-Score	Sensitivity	Specificity
Early Blight	0.97087	1	0.98522	1	0.9738
Healthy	0.96774	1	0.98361	1	0.91746
Late Blight	1	0.965	0.98219	0.965	1

Table 6: ResNet18 computational requirements

Model	Number Of Learnables	Number Of Operations	Parameter Memory (MB)	Number Of MACs	Arithmetic Intensity	FLOPs	Inference Time (ms)
ResNet18	11173251	3.627E+09	42.62257	1813562880	3436.67214	3627125760	14.50850304

ResNet18 achieves high accuracy, as suggested by all the metrics being close to 1.

The confusion matrix for AlexNet is presented in Figure 12 alongside its corresponding values in Tables 7 and 8.

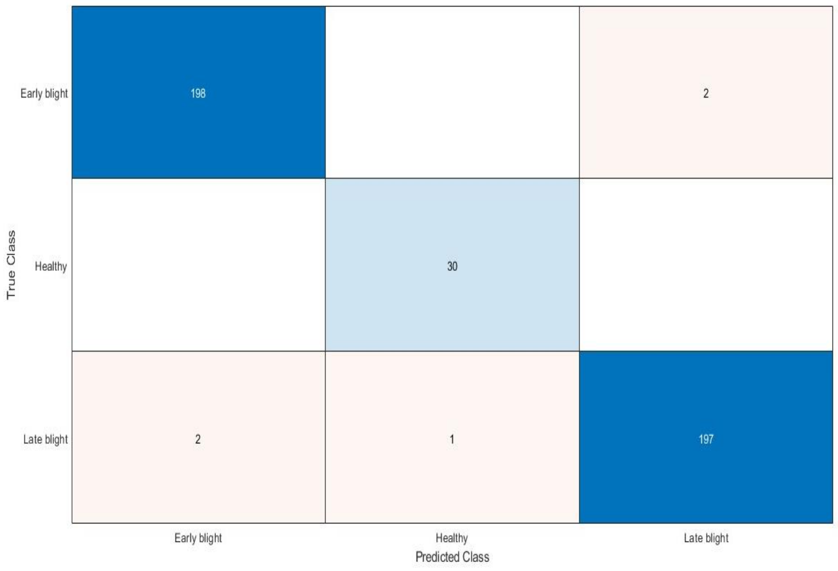


Figure 12: AlexNet Confusion matrix

Table 7: AlexNet performance metrics

	Precision	**Recall**	**F1-Score**	**Sensitivity**	**Specificity**
Early Blight	0.99	0.99	0.99	0.99	0.99127
Healthy	0.96774	1	0.98361	1	0.99747
Late Blight	0.98995	0.985	0.98747	0.985	0.9913

Table 8: AlexNet computational requirements

Model	**Number Of Learnables**	**Number Of Operations**	**Parameter Memory (MB)**	**Number Of MACs**	**Arithmetic Intensity**	**FLOPs**	**Inference Time (ms)**
AlexNet	56880417	1.441E+09	216.98157	720323006	1143.00487	1440646012	5.762584048

The confusion matrix for SqueezeNet is presented in Figure 13 alongside its corresponding values in Tables 9 and 10.

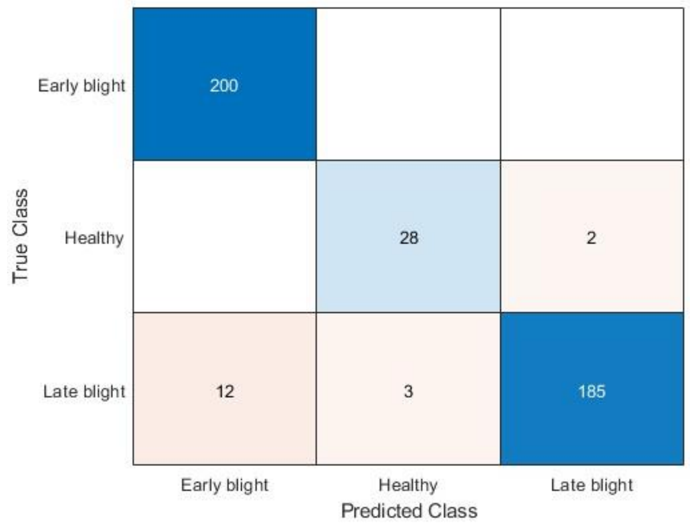


Figure 13: SqueezeNet confusion matrix

Table 9: SqueezeNet performance metrics

	Precision	Recall	F1-Score	Sensitivity	Specificity
Early Blight	0.9434	1	0.97087	1	0.94667
Healthy	0.90323	0.93333	0.91803	0.93333	0.99227
Late Blight	0.9893	0.925	0.95607	0.925	0.9913

Table 10: SqueezeNet computational requirements

Model	Number Of Learnables	Number Of Operations	Parameter Memory (MB)	Number Of MACs	Arithmetic Intensity	FLOPs	Inference Time (ms)
SqueezeNet	1823499	776671040	6.9560966	388335520	1709.30075	776671040	3.10668416

SqueezeNet, similarly, achieves high accuracy as suggested by all the metrics being close to 1

In a comparative analysis of the trained models, as indicated in figures 8 – 12. The models were compared based on their accuracy metrics and computational requirements, which are presented in Tables 1-10. Although each model shows an accuracy of close to 100%, MobileNet and ShuffleNet are the best in detecting the potato disease status using the leaves presented by the dataset. Accuracy and hardware requirements are analysed based on Figures 14 and 15.

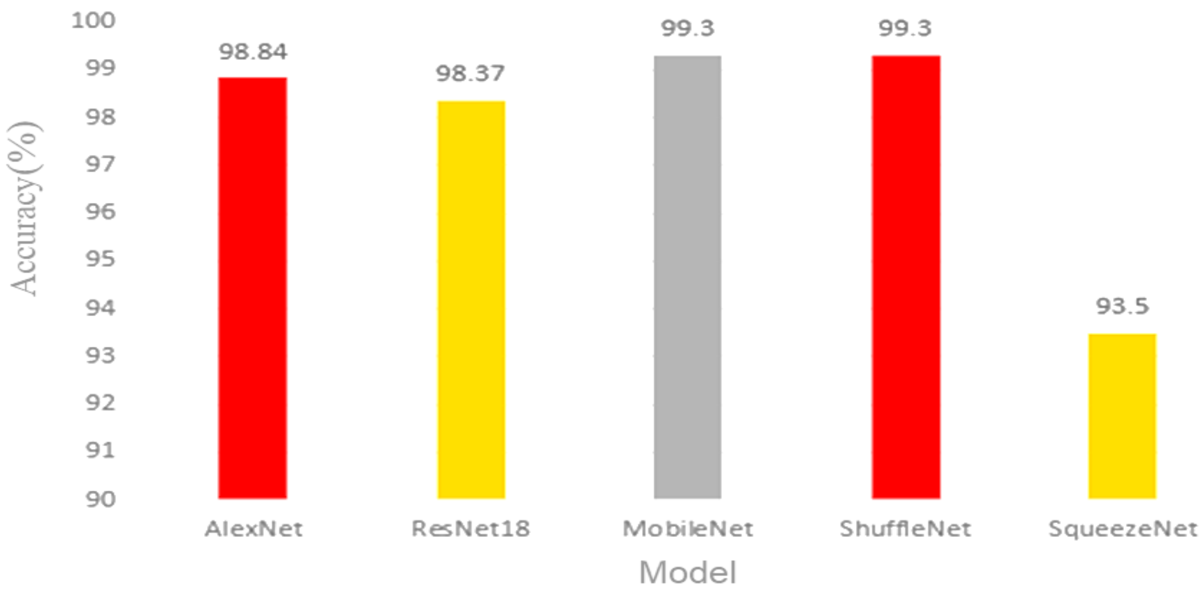


Figure 14: Accuracy comparison of trained models

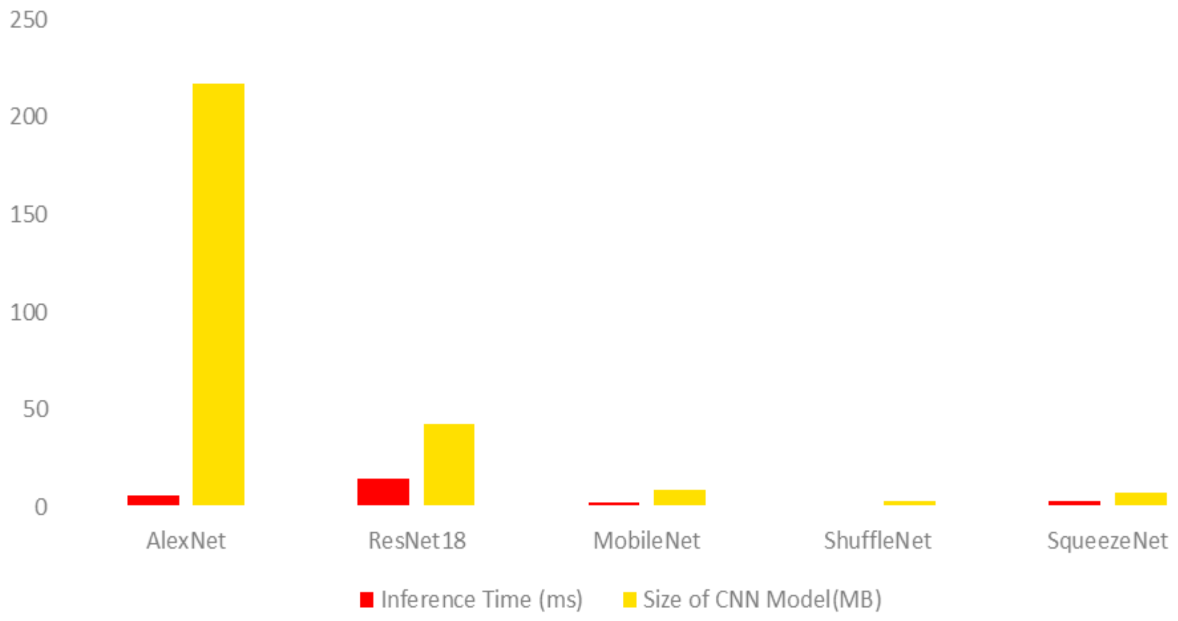


Figure 15: Hardware requirement comparison of trained models

The ShuffleNet model was selected based on its training requirement, which is the least in size among the five models, as presented in Figure 15. This is suitable for embedded devices with the highest validation accuracy (99.3%), as shown in Figure 16. Other parameters, such as the lowest inference time (0.992ms), make it suitable for real-time applications such as precision agriculture. In addition, it requires the lowest number of operations. This is regarded as lightweight and ideal for FPGA deployment as the resources are limited. It is observed that ShuffleNet model quantization conserves the hardware resources of the FPGA, and the CNN model was quantized. The quantization technique is an optimisation technique meant to reduce the size of a CNN model, making it suitable for FPGA implementation. The model parameters (i.e., weights and biases) were converted from floating point (16-bit) to fixed point (8-bit) using MATLAB.

The Effect of Quantization and quantization resulted in a reduction in classification accuracy. However, the resultant accuracy is still adequate for the application at 98.8%, as presented in Figure 16. However, it is observed that the model size did not reduce as expected; this is likely due to the lack of pruning, which is the process of removing certain layers in the model to reduce the model size.

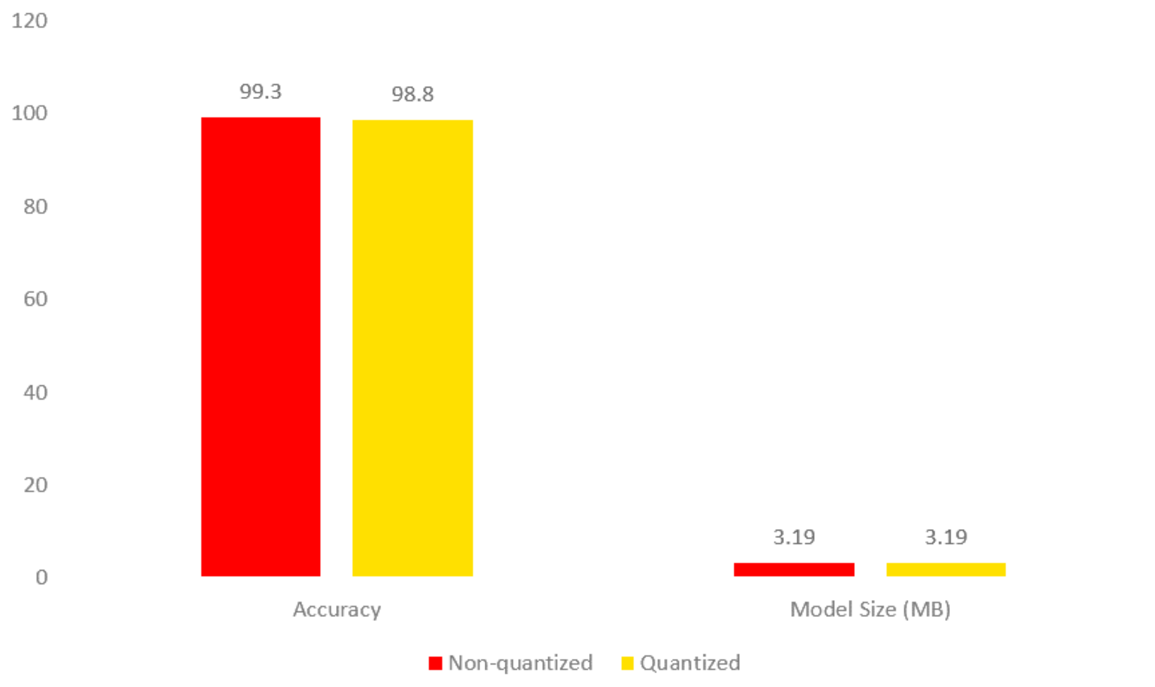


Figure 16: Effect of Quantization

The FPGA selection was successfully connected to the programming module of the simulation over SSH using ethernet, as shown in Figure 17.

JDFEWS 5 (1): 45-68, 2024
ISSN 2709-4529

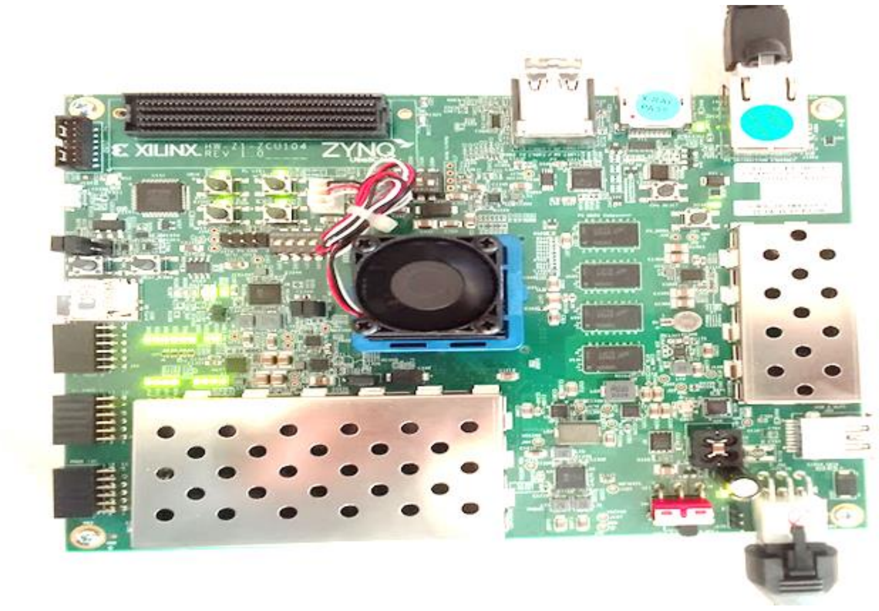


Figure 17: Ethernet connection with FPGA

In addition, the creation of a block design in Vivado HLS tool to implement the CNN using MATLAB as shown in figure 18:

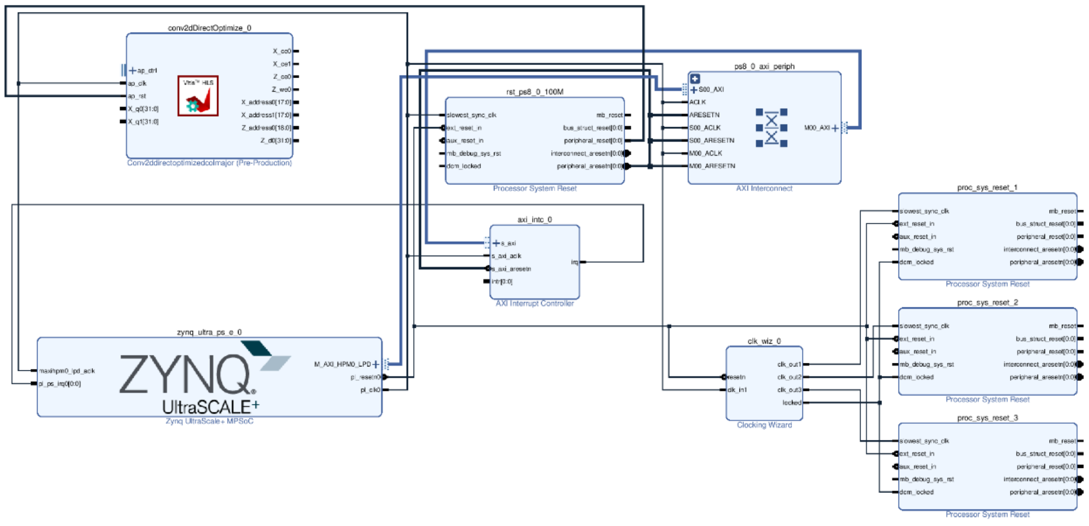
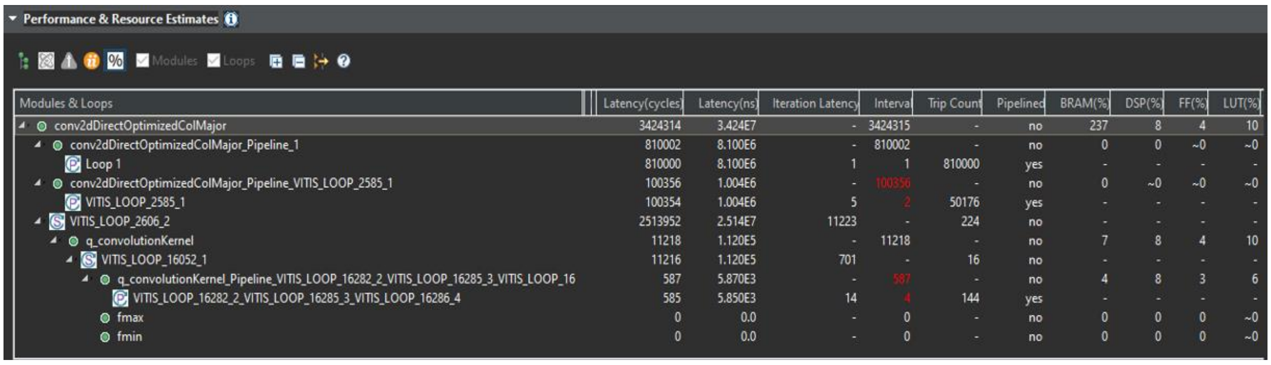


Figure 18: FPGA block design in Vivado HLS

Subsequently, Vitis HLS was used to convert the trained ShuffleNet CNN model into HDL. The HDL code was then synthesised, and the obtained resource utilization report is presented in Figure 19.

JDFEWS 5 (1): 45-68, 2024
ISSN 2709-4529



Performance & Resource Estimates

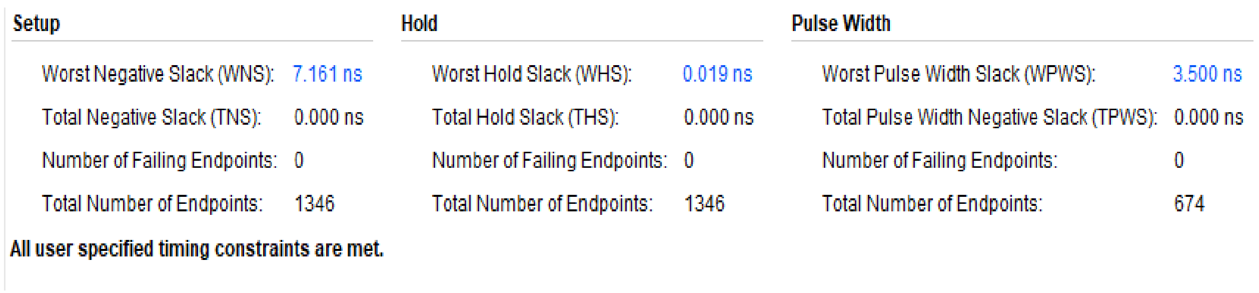
Modules & Loops	Latency(cycles)	Latency(ns)	Iteration Latency	Interval	Trip Count	Pipelined	BRAM(%)	DSP(%)	FF(%)	LUT(%)
conv2dDirectOptimizedColMajor	3424314	3.424E7	-	3424315	-	no	237	8	4	10
conv2dDirectOptimizedColMajor_Pipeline_1	810002	8.100E6	-	810002	-	no	0	0	~0	~0
Loop 1	810000	8.100E6	1	1	810000	yes	-	-	-	-
conv2dDirectOptimizedColMajor_Pipeline_VITIS_LOOP_2585_1	100356	1.004E6	-	100356	-	no	0	~0	~0	~0
VITIS_LOOP_2585_1	100354	1.004E6	5	2	50176	yes	-	-	-	-
VITIS_LOOP_2606_2	2513952	2.514E7	11223	-	224	no	-	-	-	-
q_convolutionKernel	11218	1.120E5	-	11218	-	no	7	8	4	10
VITIS_LOOP_16052_1	11216	1.120E5	701	-	16	no	-	-	-	-
q_convolutionKernel_Pipeline_VITIS_LOOP_16282_2_VITIS_LOOP_16285_3_VITIS_LOOP_16	587	5.870E3	-	587	-	no	4	8	3	6
VITIS_LOOP_16282_2_VITIS_LOOP_16285_3_VITIS_LOOP_16286_4	585	5.850E3	14	4	144	yes	-	-	-	-
fmax	0	0.0	-	0	-	no	0	0	0	~0
fmin	0	0.0	-	0	-	no	0	0	0	~0

Figure 19: Performance and FPGA resource estimates

It can be observed from the performance evaluation result that the CNN model consumed a little percentage of the FPGAs hardware resources: LUTs (10%), FF (4%), BlackRAMS (237%), and DSPs (8%); hence the CNN model was sufficiently optimised for the FPGA. Furthermore, a latency of 1.12*ns* is much faster than the value achieved in the software (MATLAB) of 0.992*ms*. In this study, five pre-trained CNN models were trained on the Plantvillage dataset on CPU+3GB GPU, using the transfer learning technique. It was observed that the training times were quite long (more than an hour on average), since the platform used was a CPU. All the CNN models achieved a high validation accuracy of more than 90%. Their performance metrics (sensitivity, recall, specificity, and F1-score) supported this performance; all were greater than 1.
Furthermore, a comparison of the CNN models showed that the ShuffleNet model was the best for FPGA-based SoC implementation because of its best performance metrics and the lowest hardware requirement. It achieved the highest validation accuracy and lowest inference time. The ShuffleNet model was then converted to HDL using Vitis HLS, and its synthesis report showed low inference time and low FPGA resource utilisation. This shows that the FPGA was more efficient than the CPU platform for CNN computation. Overall, the study compares multiple CNN models in MATLAB and selects the most suitable model for FPGA implementation based on performance metrics. This is followed by optimizing the algorithm using the quantization technique. Subsequently, the model is converted to HDL to evaluate the CNN model for an SoC platform, which has not been considered in previous studies for plant disease detection.

Hardware Performance Metrics: Timing Performance
Given that a 100 Mhz clock for the FPGA was selected, this implies that the timing limit is 10ns.The following timing metrics were obtained:

1. Worst Negative Slack (WNS) :7.161 ns
2. Worst Hold Slack (WHS): 0.019 ns
3. Worst Pulse Width Slack (WPWS): 3.500 ns



Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.161 ns	Worst Hold Slack (WHS):	0.019 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	1346	Total Number of Endpoints:	1346	Total Number of Endpoints:	674

All user specified timing constraints are met.

Figure 20: Summary of Timing Performance

Since all the values are below 10ns.This indicates that all the timing constraints are met.

Power Consumption

The power consumption obtained for the design is 3.333 W, of which 79% of the power is dynamic and 21% is static. High dynamic power correlates to the intensive requirements of CNN inference, particularly from MAC operations. Optimizing dynamic power usage can further enhance power efficiency. Nonetheless, a power consumption of 3.333W indicates that the FPGA is power efficient in inferencing the CNN model, aligning with the literature findings.

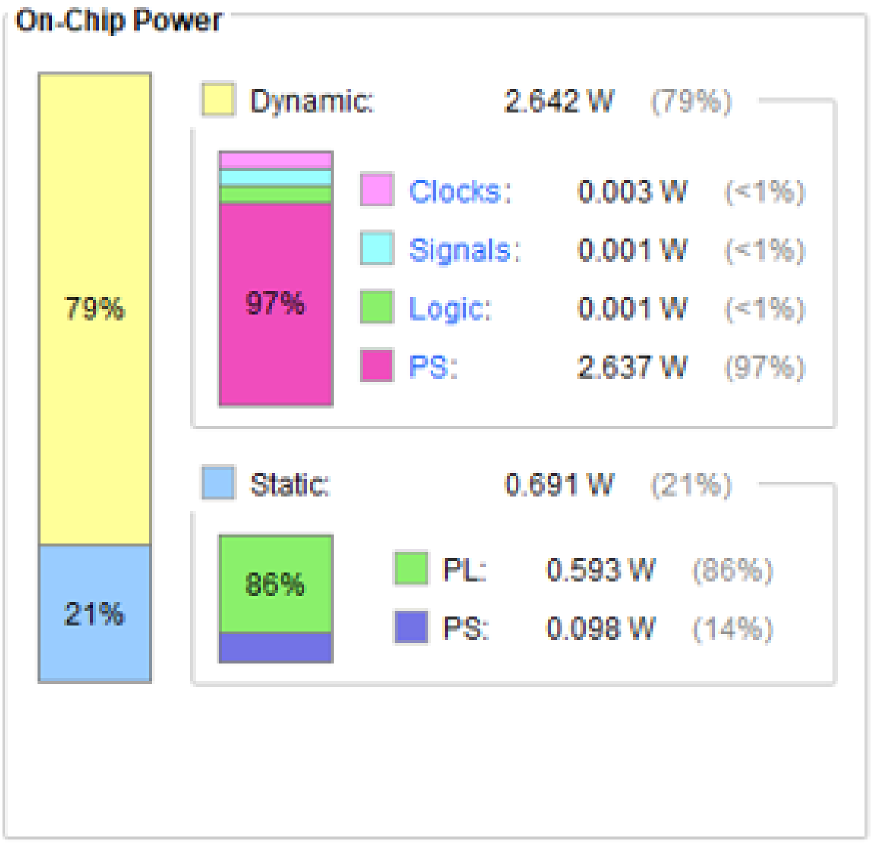


Figure 21: Power Consumption

Resource Utilisation

Table 11 shows the resource utilisation obtained from Vivado.

Table 11: Resource Utilisation

Resource	Utilization	Available	Utilization (%)
LUT	470	230400	0.2
LUTRAM	68	101760	0.07
FF	597	460800	0.13
BUFG	1	544	0.18

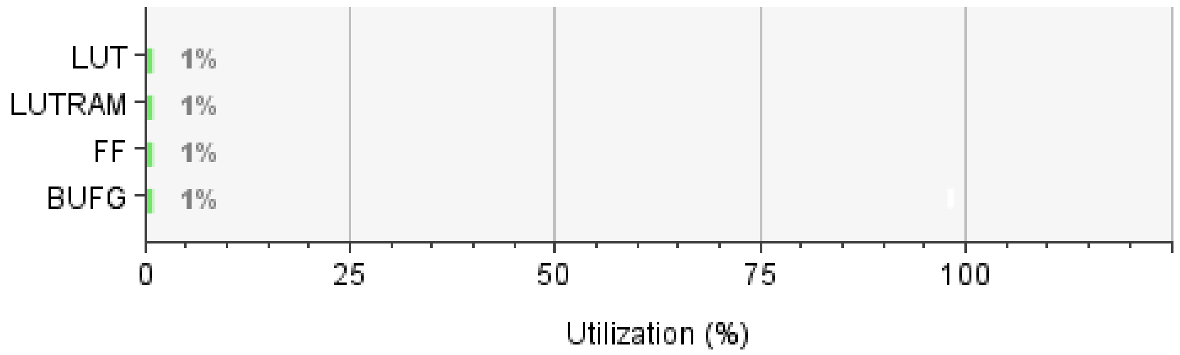


Figure 22: Percentage Resource Utilisation

LUT (Look-Up Table): A 0.2% LUT utilisation indicates that the design is not heavily reliant on combinational logic, which is typical for optimized FPGA designs where efficiency and minimizing resource usage are prioritized.
LUTRAM (Look-Up Table RAM): Similarly, a 0.07% LUTRAM ultilisation suggests efficient use of on-chip memory resources. This could be due to the effective design of data storage and handling within the CNN model.
FF (Flip-Flop): A 0.13 % utilization suggests an efficient sequential logic design. This is beneficial for maintaining low power consumption and high performance.
BUFG (Global Clock Buffer): A 0.18% BUFG indicates that the design does not require any global clock resources, which can be advantageous for minimizing clock distribution complexity and power consumption.

The resource utilization for the FPGA implementation of the CNN model (Table. 10) is very low across all key resources (LUTs, LUTRAMs, FFs, and BUFGs). This indicates a highly efficient design that conserves FPGA resources while maintaining the necessary performance for CNN inference. Such efficiency is crucial for scalability and for deploying multiple instances or additional functionality within the same FPGA. The low resource usage, combined with the power efficiency discussed earlier, underscores the effectiveness of the FPGA implementation for CNN inference.

5.0 Conclusions and Future Work

In conclusion, the study findings illustrated the effectiveness of CNNs for potato disease identification, with all the selected CNN models reaching a validation accuracy of over 90%. The ShuffleNet model was the most suitable for FPGA deployment since it achieved the highest validation accuracy and the lowest inference time, attributed to its efficient architecture. The implementation of the CNN on the Xylinx UltraScale+MPSoC ZCUQ04 board demonstrated significant efficiency in power consumption and resource utilization. The total power consumption was 3.333 W, with 79% being dynamic power, reflecting the computational demands of CNN inference. Despite this, the FPGA's power efficiency is consistent with existing literature, confirming its viability for real-time CNN tasks.
Furthermore, the design fulfilled all timing constraints with positive slack values (WNS of 7.161 ns and WHS of 0.019 ns), indicating reliable operation at the intended clock frequency of 100 MHz. This ensures that the FPGA can handle real-time inference tasks efficiently. Resource utilization was low, with only 0.20% of LUTs, 0.07% of LUTRAMs, 0.13% of flip-flops, and 0.18% of BUFGs used, indicating an optimized design that maximizes FPGA capabilities while conserving resources. The results highlight the FPGA's potential as an efficient, high-performance alternative to CPU and GPU implementations, providing a viable solution for real-time potato disease identification on platforms such as UAVs. The use of FPGAs, moreover, provides a means for real-time disease identification. Therefore, the research contributes to the existing knowledge on implementations of CNN for potato disease classification and FPGA acceleration for real-time detection.

JDFEWS 5 (1): 45-68, 2024
ISSN 2709-4529

REFERENCES

[1] Devaux, P. Kromann, and O. Ortiz, 'Potatoes for Sustainable Global Food Security', *Potato Res*, vol. 57, no. 3, pp. 185–199, 2014.

[2] M. E. Çalışkan, M. F. Yousaf, C. Yavuz, M. A. B. Zia, and S. Çalışkan, 'History, production, current trends, and future prospects', in *Potato Production Worldwide*, Elsevier, 2022, pp. 1–18. doi: 10.1016/B978-0-12-822925-5.00016-5.

[3] K. K. Bastas, 'Bacterial diseases of potato and their control', in *Potato Production Worldwide*, Elsevier, 2022, pp. 179–197. doi: 10.1016/B978-0-12-822925-5.00017-7.

[4] Namibian Agronomic Board, 'Market Intelligence Report for Potatoes', 2021. [Online]. Available: www.nab.com.na

[5] R. Kanter, H. L. Walls, M. Tak, F. Roberts, and J. Waage, 'A conceptual framework for understanding the impacts of agriculture and food system policies on nutrition and health', *Food Secur*, vol. 7, no. 4, pp. 767–777, 2015.

[6] P. M. J. P. D. N. Yashasvi Khaparde, 'PLANT CHECK: POTATO LEAF DISEASE DETECTION USING CNN MODEL', *International Journal of Engineering Applied Sciences and Technology*, vol. 8, no. 5, pp. 129–132, Sep. 2023.

[7] S. Bangari, P. Rachana, N. Gupta, P. S. Sudi, and K. K. Baniya, 'A survey on disease detection of a potato leaf using CNN', in *2022 Second International Conference on Artificial Intelligence and Smart Energy (ICAIS)*, 2022.

[8] Y. Luo, X. Cai, J. Qi, and W. Che, 'FPGA–accelerated CNN for real-time plant disease identification', *Comput Electron Agric*, vol. 207, p. 107715, 2023.

[9] C.-H. Huang, 'An FPGA-based hardware/software design using binarized neural networks for agricultural applications: A case study', *IEEE Access*, vol. 9, pp. 26523–26531, 2021.

[10] T. Saidani and R. Ghodhbani, 'Embedded plant disease recognition using Deep Plantnet on FPGA-SOC', 2022.

[11] J. Lu, L. Tan, and H. Jiang, 'Review on Convolutional Neural Network (CNN) applied to plant leaf disease classification', *Agriculture*, vol. 11, no. 8, p. 707, 2021.

[12] T. Z. J. A. A. Moshood Onifade, 'Towards application of positioning systems in the mining industry', *Int J Min Miner Eng*, vol. 15, no. 1, pp. 15–48, 2024.

[13] M. Dhanaraju, P. Chenniappan, K. Ramalingam, S. Pazhanivelan, and R. Kaliaperumal, 'Smart Farming: Internet of Things (IoT)-Based Sustainable Agriculture', *Agriculture 2022, Vol. 12, Page 1745*, vol. 12, no. 10, p. 1745, Oct. 2022, doi: 10.3390/AGRICULTURE12101745.

[14] A. Arshaghi, M. Ashourian, and L. Ghabeli, 'Potato diseases detection and classification using deep learning methods', *Multimed Tools Appl*, vol. 82, no. 4, 2023, doi: 10.1007/s11042-022-13390-1.

[15] J. Jiang *et al.*, 'Hierarchical Model Parallelism for Optimizing Inference on Many-core Processor via Decoupled 3D-CNN Structure', *ACM Transactions on Architecture and Code Optimization*, vol. 20, no. 3, 2023, doi: 10.1145/3605149.

[16] C. Hou *et al.*, 'Recognition of early blight and late blight diseases on potato leaves based on graph cut segmentation', *J Agric Food Res*, vol. 5, 2021, doi: 10.1016/j.jafr.2021.100154.

[17] R. Ghodhbani, T. Saidani, and H. Zayeni, 'Deploying deep learning networks based advanced techniques for image processing on FPGA platform', *Neural Comput Appl*, vol. 35, no. 26, 2023, doi: 10.1007/s00521-023-08718-3.

[18] R. Ghodhbani, L. Horrigue, T. Saidani, and M. Atri, 'Fast FPGA prototyping based real-time image and video processing with high-level synthesis', *International Journal of Advanced Computer Science and Applications*, no. 2, 2020, doi: 10.14569/ijacsa.2020.0110215.

[19] T. Saidani and R. Ghodhbani, 'Hardware Acceleration of Video Edge Detection with Hight Level Synthesis on the Xilinx Zynq Platform', *Engineering, Technology and Applied Science Research*, vol. 12, no. 1, 2022, doi: 10.48084/etasr.4615.

[20] S. Park and T. Suh, 'Speculative Backpropagation for CNN Parallel Training', *IEEE Access*, vol. 8, 2020, doi: 10.1109/ACCESS.2020.3040849.

[21] L. Tuggener, J. Schmidhuber, and T. Stadelmann, ‘Is it enough to optimize CNN architectures on ImageNet?’, *Front Comput Sci*, vol. 4, 2022, doi: 10.3389/fcomp.2022.1041703.

[22] A. M. Abubakar, K. A. Abdulsalam, and J. A. Adebisi, ‘Application of Artificial Neural Network for Diagnosis of Cerebrospinal Meningitis’, *Journal of Engineering Research*, vol. 24, no. 2, pp. 12–25, 2019.

[23] K. A. L. N. O. O. J Adebisi, ‘Development of an Artificial Intelligence Driven Mobile Application for Medical Diagnosis and Consultancy’, *Adeleke University Journal of Engineering and Technology*, vol. 6, no. 1, pp. 1–9, 2023.

[24] B. Zhou, X. Wang, J. Zhou, and C. Jing, ‘Trajectory Recovery Based on Interval Forward–Backward Propagation Algorithm Fusing Multi-Source Information’, *Electronics (Switzerland)*, vol. 11, no. 21, 2022, doi: 10.3390/electronics11213634.

[25] Y. Sun, B. Xue, M. Zhang, G. G. Yen, and J. Lv, ‘Automatically Designing CNN Architectures Using the Genetic Algorithm for Image Classification’, *IEEE Trans Cybern*, vol. 50, no. 9, 2020, doi: 10.1109/TCYB.2020.2983860.

[26] C. Wang, L. Li, C. Yan, Z. Wang, Y. Sun, and J. Zhang, ‘Cross-modal semantic correlation learning by Bi-CNN network’, *IET Image Process*, vol. 15, no. 14, 2021, doi: 10.1049/ipr2.12176.

[27] S. D. Okegbile and B. T. Maharaj, ‘Age of information and success probability analysis in hybrid spectrum access-based massive cognitive radio networks’, *Applied Sciences (Switzerland)*, vol. 11, no. 4, 2021, doi: 10.3390/app11041940.

[28] S. D. Okegbile, J. Cai, H. Zheng, J. Chen, and C. Yi, ‘Differentially Private Federated Multi-Task Learning Framework for Enhancing Human-To-Virtual Connectivity in Human Digital Twin’, *IEEE Journal on Selected Areas in Communications*, vol. 41, no. 11, 2023, doi: 10.1109/JSAC.2023.3310106.

[29] K. A. Abdulsalam, J. Adebisi, M. Emezirinwune, and O. Babatunde, ‘An overview and multicriteria analysis of communication technologies for smart grid applications’, *e-Prime-Advances in Electrical Engineering, Electronics and Energy*, vol. 3, p. 100121, 2023.

[30] A. John A, B. Damilola E, and B. Olubayo M, ‘A multicriteria framework for selecting information communication technology alternatives for climate change adaptation’, *Cogent Eng*, vol. 9, no. 1, p. 2119537, 2022.

[31] A. Adebisi J, B. Damilola E, and B. Olubayo M, ‘A multicriteria framework for selecting information communication technology alternatives for climate change adaptation’, *Cogent Eng*, vol. 9, no. 1, p. 2119537, 2022.

[32] J. Duchi, E. Hazan, and Y. Singer, ‘Adaptive subgradient methods for online learning and stochastic optimization’, in *COLT 2010 - The 23rd Conference on Learning Theory*, 2010.

[33] J. Duchi, E. Hazan, and Y. Singer, ‘Adaptive subgradient methods for online learning and stochastic optimization’, *Journal of Machine Learning Research*, vol. 12, 2011.