Application of Convolutional Neural Network for Identifying Cocoa Leaf Disease

Annisa Fitri Maghfiroh Harvyanti[1], Rifki Ilham Baihaki[2], Dafik[2,3], Zainur Rasyid Ridlo[2,4], and Ika Hesti Agustin[1,2(✉)]

[1] Department of Mathematics, University of Jember, Jember, Indonesia
ikahesti.fmipa@unej.ac.id
[2] PUI-PT Combinatorics and Graph, CGANT, University of Jember, Jember, Indonesia
{d.dafik,zainur.fkip}@unej.ac.id
[3] Department of Mathematics Education Postgraduate, University of Jember, Jember, Indonesia
[4] Department of Science Education, University of Jember, Jember, Indonesia

Abstract. Cocoa or Theobroma cacao L. is a plantation product that has high economic value and is very popular for its processed fruit. The large market demand for cocoa is not proportional to the low level of productivity. The main issue in cocoa plantations is the high incidence and rapid spread of disease. The most common disease is Vascular Streak Dieback (VSD). Appropriate treatment must be carried out immediately to maintain productivity. Identifying diseases based on leaf image using a Convolutional Neural Network (CNN) can simplify and speed up the detection of diseases. This study compares four CNN architectures, namely AlexNet, SqueezeNet, Darknet, and modified CNN to identify cocoa plants infected with VSD. The total number of datasets used is 1200 images, consisting of 600 images for the healthy class and the remaining 600 images for the VSD class. The best results were obtained with the DarkNet-19 model, with a test accuracy of 98.61%.

Keywords: precision agriculture · cocoa leaf disease · image classification · convolutional neural network

1 Introduction

Cocoa is one of Indonesia's mainstay agricultural commodities. Its role is quite important for the national economy, especially as a provider of employment and a source of foreign exchange. In terms of quality, Indonesian cocoa is not inferior to world cocoa because it has the advantage of not melting easily. In line with these advantages, Indonesian cocoa market opportunities are quite open, both for export and domestic demand. Nonetheless, Indonesia's cocoa production is still low due to complex problems including diseases, cocoa pod borer pests, and cocoa downstream product development [12]. One of the most common diseases in coffee plantations is Vascular Streak Disease (VSD).

VSD disease is caused by the fungus Oncobasidium theobromae (Ceratobasidiales: Ceratobasidiaceae) [8]. The first symptom to appear is usually the yellowing of the

I. H. Agustin (Ed.): ICONNSMAL 2022, AISR 177, pp. 283–304, 2023.
https://doi.org/10.2991/978-94-6463-174-6_21

leaves, especially on the second or third leaf from the tip. These leaves develop small, well-defined green spots that are scattered on a yellow background. This disease spreads very quickly and can be deadly. This is a challenge for cocoa plantations to increase cocoa productivity.

Precision agriculture is a solution required to achieve the production rate required. Applications of artificial neural networks have been widely used in precision agriculture, such as forecasting temperature, soil moisture, and relative humidity [3, 17]. An artificial neural network is one of the most important elements of machine learning and artificial intelligence. It's an adaptive system that's inspired by the human brain's structure and functions. A neural network can learn from past data to do several tasks, such as recognizing patterns, classifying data, and forecasting future events. Convolutional Neural Networks is a part of ANN which is now popularly used for image classification, segmentation, and object recognition [13]. Input data on CNN can be any data, including numeric, image, video, sound, and natural language [1, 7]. CNN is popularly used for image classification, segmentation, and object recognition [13]. CNN has been widely implemented in various fields, namely health, agriculture, web, mail services, etc.

There have been numerous studies towards the automatic classification and identification of plant diseases. In a study by Brahimi et al. [2], they classified nine tomato leaf diseases with an accuracy rate of 99.18% using AlexNet and GoogleNet. When comparing four CNN architectures to categorize three tomato leaf diseases, Baihaki et al. [14] found that VGG-16 produced the best results with a 99.67% accuracy rate. Using VGG-16 and VGG-19, Darwis et al. [4] devised a method for identifying three diseases of maize and found an average accuracy of 98.2%. Liu et al. proposed a novel architecture of CNN to identify apple leaf diseases with accuracy 97,62%. Similar research was provided by the authors employing CNN architecture to discover and recognize illnesses in maize [5, 16]. The CNN architecture is also used by the authors in other plants. Based on an article by Tugrul et al., still, no one has done research on image classification leaf plant disease for cocoa.

In this research, we apply Convolutional Neural Network as a feature extractor and classifier. We use four architectures namely AlexNet, SqueezeNet, DenseNet, and Modified-CNN. We categorize cocoa leaves into two classes, healthy class and VSD class.

2 Methods

This part describes the procedure of this research, including image acquisition, preprocessing including image augmentation and training, feature extraction, and pattern recognition using four CNN architectures, namely AlexNet, SqueezeNet, DarkNet, and modified CNN, and the last procedure is model evaluation.

A. Image Acquisition

The object research focuses on cocoa that is widely grown in South and Southeast Asia. Leaf image data is taken from cocoa plants grown at the Indonesian Coffee and Cocoa Research Institute in Jember, East Java. The leaves captured were in two classes, healthy class and VSD class. Infected leaves were observed based on the condition of the leaves based on information from literatures and farmers there. Pictures were taken

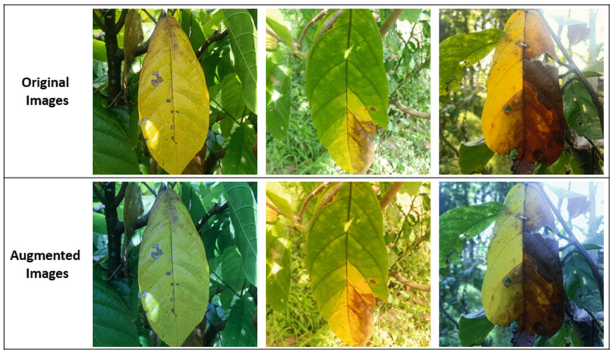

Fig. 1. Comparison of original image and augmented image.

using a smartphone camera with a 48 MP camera. All images are saved in jpg format with an image size of 3000 × 3000 pixels.

B. Data Augmentation

The initial dataset includes 343 healthy and 327 VSD-infected leaf images. The limited data we have makes it impossible to obtain an accurate and non-overfitting model. We use image augmentation to overcome this problem and improve the classification results. Data Augmentation prevents overfitting by modifying limited datasets to possess the characteristics of big data [11]. The results of the classification were improved by data augmentation. The previous study shows the effectiveness of data augmentation comes from simple transformations such as color space augmentations, flipping, and random cropping [15].

In this study, each image was augmented with a white balance color augmenter developed by Afifi et al. [10]. This augmentation method can accurately imitate realistic color constancy degradation. Existing color augmentation methods often generate unnatural colors which rarely happens. By simulating various white balance effects, white balance color augmenter increases the precision of image classification and image semantic segmentation techniques. The augmentation process produces a total of 1200 images, comprising 600 images for the healthy class and the remaining 600 for the VSD class.

C. Data Training and Testing

The data produced by the augmentation were divided by a 70:30 ratio. The number of images for training is 840 and 360 images for testing. Training data is used to generate a suitable model to test data validation. 70% of the entire data is randomly chosen for training and the remaining 30% is used for testing. Data testing is used to examine the model accuracy from the training results. The training model was devoted to generating the most accurate identification.

D. Artificial Neural Network

An artificial neural network (ANN) consists of several processing layers, such as an input layer, hidden layers, and an output layer. Each layer contains a number of nodes that take the outputs from every node in the layer below as inputs. Each neuron has a weight and threshold, which the network adjusts it depending on the error rate between the target

and actual output using a training algorithm. Any node whose output exceeds the defined threshold value is activated and begins providing data to the network's uppermost layer. Otherwise, no data is transmitted to the network's next layer.

For example, let $W = (w_{ij})$ be a weight matrix that represents the relationship between one neuron and another neuron. The network input to the target unit Y_j (with no bias to unit j) is a simple dot product of the vector $x = (x_1, x_2, \ldots, x_n)$ where n is the number of inputs and w_j is the weight value where j is the number of columns in the weight matrix. $\widehat{Y}$ becomes $\widehat{Y} = x \cdot w_j = \sum_{i=1}^{n} x_i w_{ij}$. Figure 1 illustrates a simple neural network. The bias (β) can be included in vector x by adding the component $x_0 = 1$, so that vector x becomes $x = (1, x_1, x_2, \ldots, x_n)$.

The bias is considered like any other weight, $w_{0j} = \beta_j$. The network input to the unit Y_j is given in Eq. (1).

$$\hat{Y} = \sum_{i=0}^{n} x_i w_{ij} = w_{0j} + \sum_{i=0}^{n} x_i w_{ij} = \beta_j + \sum_{i=0}^{n} x_i w_{ij} \tag{1}$$

The basic operation of a neural network is the addition and multiplication between the weights and the input signal and applying them to the activation function. These are several types of activation functions:

1. Linear activation function

   $\hat{Y} = f(x) = ax + b$, if $a = 1, b = 0$, then f is an identity function.

2. Step binary activation function with threshold θ

$$\hat{Y} = f(x) = \begin{cases} 1 \ \textit{if} \ x \geq \theta \\ 0 \ \textit{if} \ x < \theta \end{cases}$$

3. Sigmoid binary activation function

$$\hat{Y} = f(x) = \frac{1}{1 + e^{-\sigma x}}, \ \text{and} f' = \sigma f(x)\big[1 - f(x)\big].$$

4. Sigmoid bipolar activation function

$$\hat{Y} = g(x) = 2f(x) - 1 = \frac{1 - e^{-\sigma x}}{1 + e^{-\sigma x}}$$

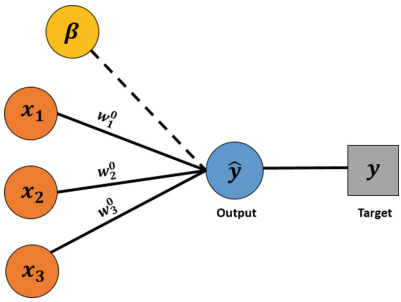

Fig. 2. A simple artificial neural network.

and $g'(x) = \frac{\sigma}{2}\left[1 + g(x)\right]\left[1 - g(x)\right]$

5. Hyperbolic tangent activation function

$$\hat{Y} = h(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}},$$

and $h'(x) = [1 + h(x)][1 - h(x)]$

Backpropagation Algorithm

Step 0. Initialize weight $w, \beta, \alpha, error\_goal$.

Step 1. Feedforward stage
$h_i = \beta + \sum_{i=0}^{n} x_i \cdot w_{i,j}$ where n is the number of inputs
Step 2. Calculate Output
$y_{in_i} = \beta + \sum_{i=0}^{n} w_{i,j} \cdot h_i$
Step 3. Activate output using sigmoid log.
$\hat{y}_m = \frac{1}{1+e^{-y_{in_m}}}$
Step 4. Calculate output error.
$\Delta k_i = (t - \hat{y}_i) \cdot y_{in_i}{}'$
Step 5. Calculate backpropagation error.
$\Delta inj_i = \sum_{i=0}^{n} w_{i,j} * \Delta k_i$
Step 6. Activate backpropagation error.
$\Delta j_i = h_i{}' * \Delta \text{inj}_i$
Step 7. Update weights and biases
$W_{new} = W_{old} + \alpha * \Delta k_i * h_i$
$\beta_{new} = \beta_{new} + \alpha * \Delta j_i + x_i$
Step 8. Checking the termination criteria (epoch reaches the maximum number or error tolerance (error goal)).
Step 9. If not, go back to Step 1.

Observation 1

Given a neural network with one hidden layer and two neurons. Figure 2 illustrates this artificial neural network. Let $x = \{x_i | i = 1, 2, \ldots, n\}$, where x is the input data and n is the number of input data. Suppose that h and β are the number of hidden layers and biases respectively. The output $(\hat{y})$ is written as $\hat{y} = \frac{1}{1+e^{-(\beta_1 + W^1(\beta_0 + W^0 x))}}$, where $[h_1; h_2] = \beta^0 + W^0 x$.

Proof

Input data $x = \{x_1, x_2, \ldots, x_n\}$, since we have one layer and two neurons, we define weights (Fig. 3):

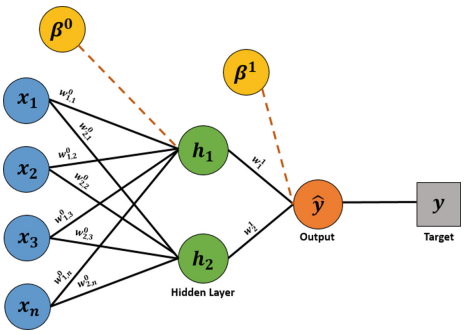

Fig. 3 Illustration of artificial neural network with 1 hidden layer.

$W^0 = [\, w^0_{1,1}\ w^0_{1,2}\ \cdots w^0_{1,n}\ ;\ w^0_{2,1}\ w^0_{2,2}\ \cdots w^0_{2,n}\ ]$, $W^1 = [\, w^1_1\ w^1_2\,]$ and bias β^0, β^1. The output from hidden layer is as follows.

$$\begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \begin{bmatrix} \beta^0_1 \\ \beta^0_2 \end{bmatrix} + \begin{bmatrix} w^0_{1,1}\ w^0_{1,2} \cdots w^0_{1,n} \\ w^0_{2,1}\ w^0_{2,1} \cdots w^0_{2,n} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \beta^0 + W^0 x$$

Next, we obtain y_{in} as follows.

$$\begin{aligned} y_{in} &= \beta^1 + \begin{bmatrix} w^1_1 \\ w^1_2 \end{bmatrix} \begin{bmatrix} h_1 \\ h_2 \end{bmatrix} = \beta^1 + \begin{bmatrix} w^1_1 \\ w^1_2 \end{bmatrix} \cdot \left(\beta^0 + W^0 x\right) \\ &= \beta^1 + W^1 \left(\beta^0 + W^0 x\right) \end{aligned}$$

We get output $(\hat{y})$ as follows.

$$\hat{y} = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-(\beta^1 + W^1(\beta^0 + W^0 x))}}$$

It concludes the proof.

Observation 2

Given a neural network architecture with two hidden layers. In each hidden layer there are two and three neurons. Figure 4 illustrates the architecture of this neural network. Let h and β be the number of hidden layers and bias, respectively. The output $(\hat{y})$ is written as $\hat{y} = \frac{1}{1+e^{-(\beta^2 + W^2(\beta^1 + W^1(\beta^0 + W^0 x)))}}$ where $h_1 = \left[h^1_1; h^1_2\right] = \beta^0 + W^0 x$ and $h_2 = \left[h^2_1; h^2_2; h^2_3\right] = \beta^1 + W^1(\beta^0 + W^0 x)$.

Proof.

Input data $x = \{x_1, x_2, \ldots, x_n\}$, since we have two hidden layers which have two neurons and three neurons respectively, we define weights:

$$W^0 = [\, w^0_{1,1}\ w^0_{1,2}\ \ldots\ w^0_{1,n}; w^0_{2,1}\ \ w^0_{2,2}\ \ldots\ w^0_{2,n}\ \ ] W^1 = [\, w^1_{1,1}\ w^1_{1,2}\,;\ w^1_{2,1}\ w^1_{2,2}\,;\ w^1_{3,1}\ w^1_{3,2}\,]$$

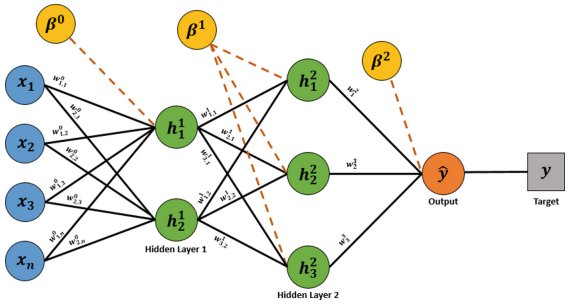

Fig. 4 Illustration of artificial neural network with 2 hidden layers.

$W^2 = [w_1^2; w_2^2; w_3^2]$ and bias β^0, β^1, β^2.
The output from hidden layer 1 is as follows:

$$\begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix} = \begin{bmatrix} \beta_1^0 \\ \beta_2^0 \end{bmatrix} + \begin{bmatrix} w_{1,1}^0 & w_{1,2}^0 \cdots w_{1,n}^0 \\ w_{2,1}^0 & w_{2,1}^0 \cdots w_{2,n}^0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} = \beta^0 + W^0 x$$

The output of hidden layer 2 is as follows:

$$\begin{bmatrix} h_1^2 \\ h_2^2 \\ h_3^2 \end{bmatrix} = \begin{bmatrix} \beta_1^1 \\ \beta_2^1 \\ \beta_3^1 \end{bmatrix} + \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & \cdots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,1}^1 & \cdots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & \cdots & w_{3,n}^1 \end{bmatrix} \begin{bmatrix} h_1^1 \\ h_2^1 \end{bmatrix}$$

$$\begin{bmatrix} h_1^2 \\ h_2^2 \\ h_3^2 \end{bmatrix} = \begin{bmatrix} \beta_1^1 \\ \beta_2^1 \\ \beta_3^1 \end{bmatrix} + \begin{bmatrix} w_{1,1}^1 & w_{1,2}^1 & \cdots & w_{1,n}^1 \\ w_{2,1}^1 & w_{2,1}^1 & \cdots & w_{2,n}^1 \\ w_{3,1}^1 & w_{3,2}^1 & \cdots & w_{3,n}^1 \end{bmatrix} \cdot (\beta^0 + W^0 x)$$

$$\left[h_2^2\right] = \beta^1 + W^1(\beta^0 + W^0 x)$$

Next, we obtain y_{in} as follows.

$$y_{in} = \beta^2 + \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \end{bmatrix} \begin{bmatrix} h_1^2 \\ h_2^2 \\ h_3^2 \end{bmatrix}$$

$$y_{in} = \beta^2 + \begin{bmatrix} w_1^2 \\ w_2^2 \\ w_3^2 \end{bmatrix} \cdot \left(\beta^1 + W^1\left(\beta^0 + W^0 x\right)\right)$$

$$y_{in} = \beta^2 + W^2\left(\beta^1 + W^1\left(\beta^0 + W^0 x\right)\right)$$

The output result ($\hat{y}$) is as follows.

$$\hat{y} = \frac{1}{1 + e^{-y_{in}}} = \frac{1}{1 + e^{-(\beta^2 + W^2(\beta^1 + W^1(\beta^0 + W^0 x)))}}$$

It concludes the proof.

E. Convolutional Neural Network

Convolutional Neural Network is a part of a deep neural network, which is a type of ANN that is generally used in image processing and recognition. In general, there are three layers in CNN, an input layer, one or some hidden layers, and an output layer. The operation in hidden layer changes data to learn special features in the data. The three most common layers in CNN are convolution layer, activation layer, and pooling layer.

Convolution runs a series of convolutional filters through the input images, activating different aspects of the images with each filter.

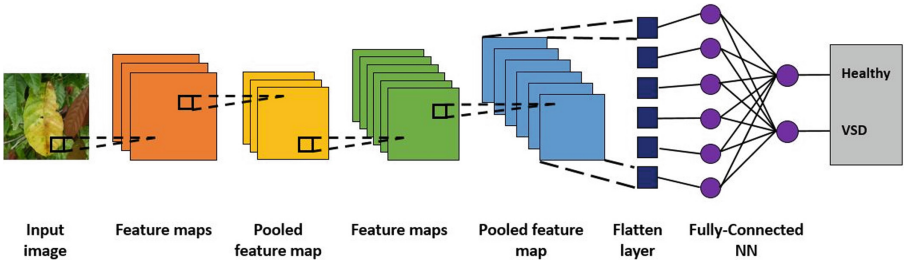

Fig. 5 Architecture of convolutional neural network.

Rectified linear unit (ReLU) maintains positive values while translating negative values to zero, enables quicker and more efficient training. Due to the fact that only the activated features are carried over to the following layer, this is frequently referred to as activation.

Pooling reduces the number of parameters the network needs to learn by conducting nonlinear down sampling on the output.

Each layer learns to recognize various features as these operations are repeated over tens or hundreds of layers.

F. CNN Architecture

AlexNet

AlexNet earned first place in the 2012 ImageNet Large Scale Visual Recognition Challenge (ILSVRC), a large-scale image classification competition in 2012. This architecture consists of 5 convolution layers, 3 pooling layers, 2 dropout layers, and 3 fully connected layers. AlexNet was designed by Alex Krizhevsky, Geoffrey Hinton, and Ilya Sutskever who are members of the SuperVision group.

SqueezeNet

SqueezeNet is an improved architecture of AlexNet. It has 18 layers and a 227x227 image input size. SqueezeNet reduces the size of the activation map (squeeze) by changing the convolution array from 3x3 to 1x1 and using fewer filters. With 50 times fewer parameters, SqueezeNet achieves AlexNet-level accuracy on ImageNet [4].

DarkNet-19

DarkNet-19 is a convolutional neural network that has 19 layers deep. The pretrained network can classify images into 1000 object categories on ImageNet. This network has a 256x256 image input size.

Modified CNN

This architecture is developed by Baihaki et al. [15] to classify CT-Scan images of the lungs of patients with COVID-19 with an accuracy rate of 91% for testing and 100% for training. This architecture has 8 layers deep with 2 convolutional layers, two activation layers, one softmax layer, and one classification layer. It uses adam as the optimizer.

G. Simulation of CNN

CNN is specifically designed to process pixel data and visual images. Neurons that compose CNN perform the roles of activation, bias, and weight. Additionally, neurons are organized in the convolutional layer to create a filter with length and height (pixels). In order to get new representative information from the multiplication of the picture component with the applied filter, CNN applies a convolution kernel (filter) to an image. The following is the simulation of CNN workflows.

- Splitting Image.

  Every image has three color channels, namely RGB (red, green, blue). The image of size 256 × 256 pixels is actually a multidimensional array with a size of 256 × 256 × 3 (3 is the number of color channels). The array is stored and read as a representation of an image. Let's take a small size 5 × 5 pixels of an image as an example to simulate the process of CNN (Fig. 6).

- Feature Extraction

  Numbers that represent the image are features that result from the encoding of an image. The feature extraction layer composes of Convolutional Layer and the Pooling Layer (Fig. 7).

- Zero Padding

Zero padding is the addition of zero rows and columns on each side of the input. This is done in order to manipulate the output dimensions of the convolutional layer (feature map) so that more features are properly extracted (Fig. 8).

- Convolution Process

Given a certain sized filter that will be shifted by a stride to the entire part of the image. Convolution performs a "dot" operation between the input and the value of the

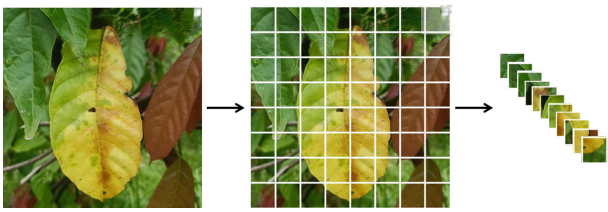

Fig. 6 Splitting input image.

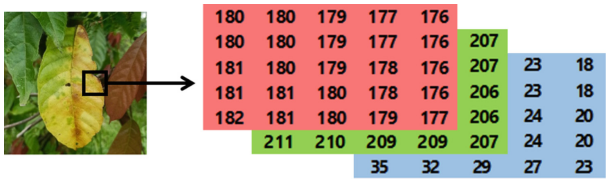

Fig. 7 Color channel on input image.

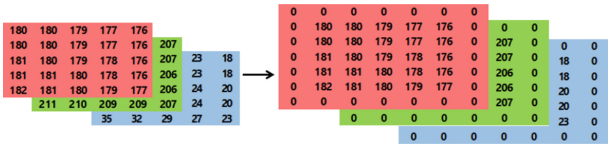

Fig. 8 Zero padding on input image.

filter to produce an activation map or feature map. In the example, a 3×3 filter is taken with a stride of 1. The following is a simulation of convolution process.

Operate convolution process in the red channel:

$$\begin{aligned} I_{red} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 180 & 180 \\ 0 & 180 & 180 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ -1 & -1 & 0 \\ 1 & 1 & 1 \end{bmatrix} \\ &= (0 * 1) + (0 * 0) + \ldots + (180 * 1) \\ &= 180 \end{aligned}$$

Convolution process in the green channel:

$$\begin{aligned} I_{green} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 210 & 209 \\ 0 & 210 & 209 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & -1 \end{bmatrix} \\ &= (0 * -1) + (0 * 0) + \ldots + (209 * -1) \\ &= 210 \end{aligned}$$

Convolution process in the blue channel:

$$\begin{aligned} I_{blue} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 34 & 31 \\ 0 & 34 & 31 \end{bmatrix} * \begin{bmatrix} -1 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix} \\ &= (0 * -1) + (0 * 0) + \ldots + (31 * 0) \\ &= -3 \end{aligned}$$

Let bias $\beta = 1$, the feature map values results are as follows:

$$\begin{aligned} c_1 &= I_{red} + I_{green} + I_{blue} + \beta \\ &= 180 + 210 - 3 + 1 \\ &= 388 \end{aligned}$$

The process of convolution operation is continued by shifting the filter one pixel to the right.

Convolution process in the red channel:

$$I_{red} = \begin{bmatrix} 0 & 0 & 0 \\ 180 & 180 & 179 \\ 180 & 180 & 179 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ -1 & -1 & 0 \\ 1 & 1 & 1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 0) + \ldots + (179 * 1)$$
$$= 179$$

Convolution process in the green channel:

$$I_{green} = \begin{bmatrix} 0 & 0 & 0 \\ 210 & 209 & 208 \\ 210 & 209 & 208 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & -1 \end{bmatrix}$$
$$= (0 * -1) + (0 * 0) + \ldots + (208 * -1)$$
$$= -1$$

Convolution process in the blue channel:

$$I_{blue} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 34 & 31 \\ 0 & 34 & 31 \end{bmatrix} * \begin{bmatrix} -1 & 0 & -1 \\ 0 & 0 & 1 \\ 0 & -1 & 0 \end{bmatrix}$$
$$= (0 * -1) + (0 * 0) + \ldots + (31 * 0)$$
$$= -3$$

Then, the feature map values results are as follows:

$$c_2 = I_{red} + I_{green} + I_{blue} + \beta$$
$$= 179 - 1 - 3 + 1$$
$$= 176$$

A series of convolution processes have been carried out, then a feature map is obtained as follows:

$$C = \begin{bmatrix} c_1 & c_2 & \cdots & c_5 \\ c_6 & c_7 & \cdots & c_{10} \\ \vdots & \vdots & \ddots & \vdots \\ c_{21} & c_{22} & \cdots & c_{25} \end{bmatrix}$$

$$C = \begin{bmatrix} 388 & 176 & 172 & 171 & -17 \\ 358 & 85 & 89 & 97 & -72 \\ 357 & 84 & 268 & 98 & -75 \\ 359 & 87 & 90 & 94 & -77 \\ 240 & -6 & 0 & 3 & -203 \end{bmatrix}$$

- Activation Layer (ReLU)

The reLU (Rectified Linear Unit) activation function is one of the most frequently used to activate feature maps. The reLU function is used to represent the output value of the neuron as 0 if the input is negative. If the input is positive, then the value is retained.

$$C(\sigma) = \begin{cases} \sigma, \text{if} \sigma \geq 0 \\ 0, \text{if} \sigma < 0 \end{cases}$$

$$C = \begin{bmatrix} 388 & 176 & 172 & 171 & 0 \\ 358 & 85 & 89 & 97 & 0 \\ 357 & 84 & 268 & 98 & 0 \\ 359 & 87 & 90 & 94 & 0 \\ 240 & 0 & 0 & 3 & 0 \end{bmatrix}$$

- Pooling Layer (Max Pooling)

In the pooling layer, there is a certain filter and stride that is shifted throughout the feature map area. The most common pooling methods are Average Pooling and Max Pooling. The goal is to reduce the dimensions of the feature map (down sampling), thereby speeding up processing because there are fewer parameters to update and overcome overfitting. The Max Pooling process is illustrated as follows:

$$m_1 = \begin{bmatrix} 388 & 176 & 172 \\ 358 & 85 & 89 \\ 357 & 84 & 268 \end{bmatrix}$$

$$\max(m_1) = 388$$

The next filter shifts 1 pixel to the right, the results of the MaxPooling process on the filter are illustrated below.

$$m_2 = \begin{bmatrix} 176 & 172 & 171 \\ 85 & 89 & 97 \\ 84 & 268 & 98 \end{bmatrix}$$

$$\max(m_2) = 268$$

After going through a series of MaxPooling processes, the following feature map is obtained.

$$C = \begin{bmatrix} 388 & 268 & 268 \\ 359 & 268 & 268 \\ 359 & 268 & 268 \end{bmatrix}$$

The process will return to the convolution operation if there is still a Convolution Layer after the MaxPooling Layer.

Flattening

This is the last process in the feature extraction layer. Flattening or reshaping a feature map means converting it from a matrix form to a vector.

$$C = [388; 268; 268; 359; 268; 268; 359; 268; 268]^T$$

These vectors must be normalized first before become input to Fully-Connected Neural Networks (FCNN).

Normalization

We normalize vector C with the following formula.

$$x_i = \frac{a + (C_i - \max(C)) \times (b - a)}{\max(C) - \min(C)}$$

where $a = 0.1; b = 0.9$

We obtain the input vector for FCNN as follows.

$$x = [0.9; 0, 1; 0, 1; 0, 707; 0, 1; 0, 1; 0, 707; 0, 1; 0, 1]^T$$

These vectors then become input to Fully-Connected Neural Networks (FCNN). Some of the parameters we used in FCNN are weights (W^0andW^1), bias (β^0andβ^1), learning rate $\alpha = 0.01$, and target (t).

- Fully-Connected Neural Network (FCNN)

The fully connected layer is like an artificial neural network, where all the activity neurons from the previous layer are connected to the neurons in the next layer. FCNN aims to process data so that it can be classified, and it is usually used in the multi-layer perceptron method (Fig. 9).

Some of the parameters used in FCNN are weights (w_1, w_2, w_3, w_4), bias (v), learning rate $\alpha = 0.01$, and target (t).

$$W^0 = \begin{bmatrix} 0.2\ 0, 1\ 0, 1\ 0, 4\ 0, 2\ 0, 2\ 0, 3\ 0, 1\ 0, 1 \\ 0, 1\ 0, 3\ 0, 2\ 0, 2\ 0, 1\ 0, 4\ 0, 1\ 0, 1\ 0, 2 \end{bmatrix}$$

$$W^1 = [0, 2; 0, 1]$$

$$\beta^0 = [0.1; 0, 1]$$

$$\beta^1 = [0.1; 0, 1]$$

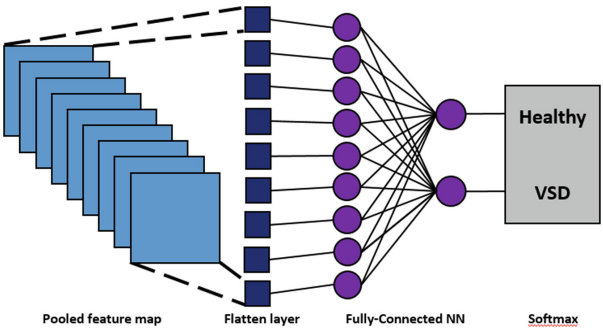

Fig. 9 Illustration of Fully Connected Neural Network.

$$t = \begin{bmatrix} 1 & 1 \end{bmatrix}$$

The first step in FCNN is feedforward neuron one as follow.

$$h_{1,1}^{1} = \beta_{1}^{0} + x_1 * w_{1,1}^{0} = 0.1 + 0.9 * 0.2 = 0.2800$$

$$h_{1,2}^{1} = \beta_{1}^{0} + x_2 * w_{1,2}^{0} = 0.1 + 0.1 * 0.1 = 0.1100$$

$$\vdots$$

$$h_{1,9}^{1} = \beta_{1}^{0} + x_9 * w_{1,9}^{0} = 0.1 + 0.1 * 0.1 = 0.1100$$

$$h_{1}^{1} = [0.2800\ 0.1100\ 0.1100\ 0.3828\ 0.1200$$

$$0.1200\ \ 0.3121\ 0.1100\ 0.1100\ ]$$

Next is feedforward neuron two as follow.

$$h_{2,1}^{1} = \beta_{2}^{0} + x_1 * w_{2,1}^{0} = 0.1 + 0.9 * 0.1 = 0.1900$$

$$h_{2,2}^{1} = \beta_{2}^{0} + x_2 * w_{2,2}^{0} = 0.1 + 0.1 * 0.3 = 0.1300$$

$$\vdots$$

$$h_{2,9}^{1} = \beta_{2}^{0} + x_9 * w_{2,9}^{0} = 0.1 + 0.1 * 0.2 = 0.1200$$

$$h_{2,9}^{1} = [0.1900\ 0.1300\ 0.1200\ 0.2414\ 0.1100$$

$$0.1400\ \ 0.1707\ 0.1100\ 0.1200\ ]$$

Next is to calculate the output (y)

$$yin_1 = \beta_{1}^{1} + w_{1}^{1} * h_{1,1}^{1} + w_{1}^{1} * h_{1,2}^{1} + \cdots + w_{1}^{1} * h_{1,9}^{1}$$

$$yin_1 = 0.1 + 0.2 * 0.2800 + 0.2 * 0.1100 + \cdots + 0.1 * 0.1100$$

$$yin_1 = 0.4310$$

$$yin_2 = \beta_{2}^{1} + w_{2}^{1} * h_{2,1}^{1} + w_{2}^{1} * h_{2,2}^{1} + \cdots + w_{2}^{1} * h_{2,9}^{1}$$

$$yin_2 = 0.1 + 0.1 * 0.1900 + 0.1 * 0.1300 + \cdots + 0.1 * 0.1210$$

$$yin_2 = 0.2332$$

Then activate each neuron output. The activation function used is the sigmoid log.

$$y_m = \frac{1}{1 + e^{-yin_m}}$$

$$y_1 = 0.6061$$

$$y_2 = 0.5580$$

After obtaining the next output, we can calculate the output error.

$$\delta k_1 = (t_1 - y_1)^2$$

$$\Delta k_1 = (1 - 0.6061)^2$$

$$\Delta k_1 = 0.1552$$

$$\delta k_2 = (t_2 - y_2)^2$$

$$\Delta k_2 = (1 - 0.5580)^2$$

$$\Delta k_2 = 0.1953$$

$$MSE = \frac{((t_1 - y_1) + (t_2 - y_2))^2}{2} = 0.3493$$

Then calculate the backpropagation error between output layer and hidden layer:

$$\delta j_1^0 = \beta_1^0 + w_1^1 * \delta k_1$$

$$\delta j_1^0 = 0.1 + 0.2 * 0.1552 = 0.1310$$

$$\delta j_2^0 = \beta_2^0 + w_2^1 * \delta k_2$$

$$\delta j_2^0 = 0.1 + 0.1 * 0.1953 = 0.1195$$

$$\delta j_2^0$$

Then calculate the backpropagation error between hidden layer and input layer:

$$\delta j_1^1 = w_{1,1}^0 * \delta k_1 + w_{1,2}^0 * \delta k_1 + w_{1,3}^0 * \delta k_1 + \cdots + w_{1,9}^0 * \delta k_1$$

$$\delta j_1^1 = 0.2638$$

$$\delta j_2^1 = w_{2,1}^0 * \delta k_2 + w_{2,2}^0 * \delta k_2 + w_{2,3}^0 * \delta k_2 + \cdots + w_{2,9}^0 * \delta k_2$$

$$\delta j_2^1 = 0.3321$$

The final step is to update the weights and biases as follows:

- The weights between input layer and hidden layer

$$w\_new_{1,1}^0 = w_{1,1}^0 + \alpha * \delta k_1 * h_1^1$$

$$w\_new_{1,1}^0 = [0.2004; 0.1001; 0.1001; 0.4005; 0.2002; 0.2002;$$

$w\_new_{1,1}^0 = 0.3004; 0.1001; 0.1001]$

$$w\_new_{2,1}^0 = w_{2,1}^0 + \alpha * \delta k_2 * h_2^1$$

$$w\_new_{1,1}^0 = [0.1002; 0.3002; 0.2001; 0.2003; 01001; 0.4002;$$

$w\_new_{1,1}^0 = 0.1002; 0.1001; 0.2001]$

$$w\_new_{2,1}^0 = \begin{bmatrix} w\_new_{1,1}^0 \\ w\_new_{2,1}^0 \end{bmatrix}$$

$$= \begin{bmatrix} 0.20040.10010.10010.40050.20020.20020.30040.10010.1001 \\ 0.10020.30020.20010.2003010010.40020.10020.10010.2001 \end{bmatrix}^T$$

- The weights between hidden layer and output layer

$$w\_new_1^1 = w_1^1 + \alpha * \delta k_1 = 0.2030$$

$$w\_new_2^1 = w_2^1 + \alpha * \delta k_2 = 0.1035$$

$$w\_new^1 = \begin{bmatrix} w\_new_1^1 \\ w\_new_2^1 \end{bmatrix} = \begin{bmatrix} 0.2030 \\ 0.1035 \end{bmatrix}$$

- Bias between input layer and hidden layer

$$\beta\_new_1^0 = \beta_1^0 + \alpha \cdot \delta \mathrm{j}_1 = 0.1013$$

$$\beta\_new_2^0 = \beta_1^0 + \alpha \cdot \delta \mathrm{j}_2 = 0.1012$$

$$\beta\_new^0 = \left[\beta\_new_1^0; \beta\_new_2^0; \right] = [0.1119; 0.1131; 0.1146]$$

- Bias between hidden layer and output layer

$$\beta\_new_1^1 = \beta_1^1 + \alpha \cdot \delta j_1 = 0.1013$$

The process is repeated to the feedforward stage until epoch reaches the maximum number or error tolerance (error_goal). The output value of the neural network is then used as input to the Softmax Layer.

Softmax Layer

Softmax layer is implemented just before the output layer in neural network. In this layer, there will be as many probabilities as the output class we specify. An illustration of the softmax process is shown in Fig. 10 and 11. The output value (logits) is processed using the softmax function. This layer must have the same number of nodes as the output layer or the number of classes, in classification problem. The softmax function is often used as the final activation function of a neural network to normalize the network output to a probability distribution over predicted output class.

H. Confusion Matrix

A table called a confusion matrix is used to describe how well a classification system performs. The output of a classification algorithm is shown and summarized in a confusion matrix.

This confusion matrix can be used to evaluate the performance of a model, such as accuracy, precision, and recall. Accuration shows the correctness rate of the model. Precision compares the amount of data that is classified as true with data that is classified as true and data that is classified as false from the true class. Recall compares the data classified as true with data classified as true and data classified as false not from the true class. Those are calculated using Eqs. (1), (2), and (3).

$$Accuration = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{1}$$

$$Precission = \frac{TP}{TP + FP} \times 100\% \tag{2}$$

$$Recall = \frac{TP}{TP + FN} \times 100\% \tag{3}$$

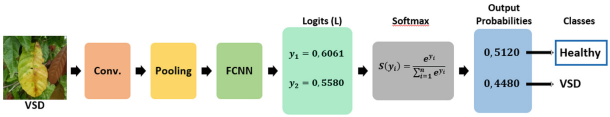

Fig. 10 Illustration of the Softmax process with output that does not matches the target.

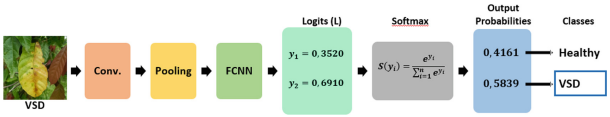

Fig. 11 Illustration of the Softmax process with output that matches the target.

3 Design and Experiment

In this study, the data set used was taken from cocoa plants grown at the Indonesian Coffee and Cocoa Research Institute in Jember, East Java. The leaves captured were healthy and infected with VSD. There are two data classes in this data set, namely healthy class and VSD class. The distribution of the data set used for network training and testing can be seen in Table 2.

The data set consists of 1200 images, with 840 images used as training and the remaining 360 images used as testing. The training stage aims to train the neural network to recognize the given image input. The output of network training is training accuracy and training model. In the testing phase, this model is used to recognize new data.

4 Results and Discussion

A. Metrics and Envirenment Setup

In this study, we use Adam optimizer as an algorithm for updating weights and biases in training stage. The number of epochs that we use in this study is 100 epochs. Other parameters we used at the training stage are the learning rate at 0,0001 and 10 mini batch size (Table 1).

The computer used in this study used Matlab 2022a software, DESKTOP-PJ6S0T6 and some hardware: Intel(R) Celeron(R) N4000 CPU @ 1.10GHz 1.10 GHz with 4 GB of memory.

B. The Evaluation of Training Stage

In this study we used CNN Transfer Learning, which is a model that has been trained using another data set. In this study, we use Alexnet, SqueezeNet, DarkNet-19, and Modified CNN. Each of these models is identical to the original architecture, but without the fully-connected layer.

Table 1. Confusion MATRIX

	True	**False**
True	True Positive (TP)	False Positive (FP)
False	False Negative (FN)	True Negative (TN)

Table 2. Data Set

Class	**Data Set**	
	Training	***Testing***
VSD	420 Images	180 Images
Healthy	420 Images	180 Images
Total	**840 Images**	**360 Images**

Table 3. Training and Testing Accuracy

Model CNN	Train Accuracy	Time Train	Test Accuracy
Modified CNN	100%	6970 s	94,17%
SqueezeNet	97,62%	8615 s	97,5%
Alexnet	98,69%	7846 s	97,78%
DarkNet-19	99,76%	8460 s	98,61%

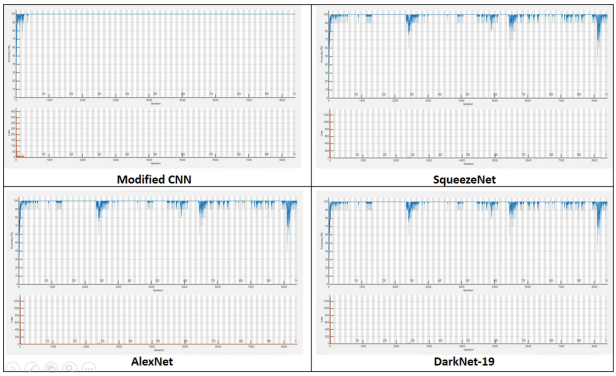

Fig. 12 Comparison of Plot Training on Four CNN Models.

Each CNN model generates training and loss plots in training stage. We can observe the training accuracy and its loss as the epoch goes on. The higher the accuracy value, the lower the loss value of the model (Fig. 12).

C. The Evaluation of Testing Stage

The trained model is tested on new data, which is called test data. In this test, we can observe the level of accuracy of the model we have. We use the resulting test accuracy value as a benchmark for model accuracy. Based on Table 3, the best CNN model is produced by DarkNet-19 with a test accuracy of 98,61%. It is followed by AlexNet 97,78%, SqueezeNet 97,5%, and Modified CNN 94,17%. Each of these models ithen measured its performance with a confusion matrix. The comparison of the confusion matrix in each CNN model is shown in Fig. 13.

In Fig. 13 the smallest number of errors is owned by DarkNet-19. This is why the accuracy of the DarkNet-19 test reaches 98.61%. The model that generates the most errors is Modified CNN.

We calculate accuracy (Eq. 1), precision (Eq. 2), and recall (Eq. 3). In this study, we use accuracy, precision, and recall as benchmarks for the performance of the classification model. Figure 13 shows the performance of the four classification models.

Based on the confusion matrix, the best classification model is produced by DarkNet-19, with 98,61% accuracy, 98,60% precision, and 98,62% recall. Accuracy shows how much the accuracy of the model is, which means that the greater the accuracy, the smaller the error. Precision indicates how much data belonging to one class is misclassified as

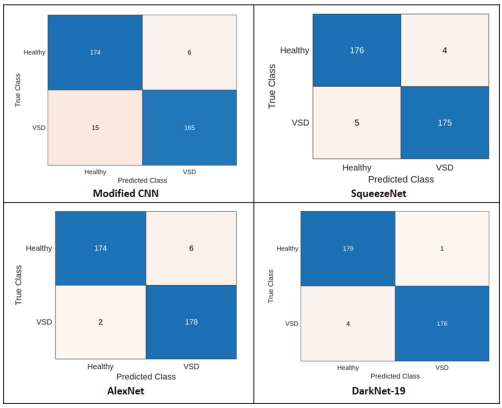

Fig. 13 The Confusion Matrix of Four CNN Models.

another class. The greater the precision value, the smaller the error made. Meanwhile, recall shows how many other classes are incorrectly classified as a class. The greater the recall value indicates the smaller the error made by the system.

5 Conclusions and Future Works

Agroindustry is a very important aspect of supporting the community's economy. Cocoa is an agroindustrial product that is an export commodity. However, the level of cacao production is low due to the high spread of a deadly disease, namely VSD. For this reason, it is necessary to create a system model that can identify diseases quickly and efficiently so that treatment can be carried out immediately. Nowadays, artificial neural network is widely used to do precision agriculture. In this study, we implemented deep learning, especially convolutional neural networks to classify healthy and VSD-infected cocoa plants. For comparison, we used four CNN models to see which one was more effective in solving our problem. We use four models, namely Modified CNN, AlexNet, SqueezeNet, and DarkNet-19. The best results were obtained using the DarkNet-19 model, with a test accuracy of 98.61%.

Acknowledgment. This project is funded by PUI-PT Combinatorics and Graph, CGANT, University of Jember 2023.

References

1. O. Abdel-Hamid, A.R. Mohamed, H. Jiang, L. Deng, G. Penn, and D. Yu, "Convolutional Neural Networks for Speech Recognition" IEEE/ACM Transactions on Audio, Speech, and Language Processing, vol. 22, no. 10, October 2014, pp. 1533–1545
2. M. Brahimi, B. Kamel, A. Moussaoui, "Deep Learning for Tomato Diseases: Classification and Symptoms Visualization" Applied Artificial Intelligence, 6 April 2017, 31, pp 1-17

3. Dafik, Z. R. Ridlo, I. H. Agustin, R. I. Baihaki, F. G. Febrinanto, R. Nisviasari, Suhardi, and A. Riski, "The Implementation of Artificial Neural Networks and Resolving Efficient Dominating Set for Time Series Forecasting on Soil Moisture to Advance the Automatic Irrigation System on Vertical Farming" in press
4. A. Darwish, D. Ezzat, A. E. Hassanien, "An Optimized Model Based On Convolutional Neural Networks And Orthogonal Learning Particle Swarm Optimization Algorithm For Plant Diseases Diagnosis" Swarm and Evolutionary Computation, vol. 52, 2020
5. C. DeChant, T. Wiesner-Hanks, S. Chen, E. L. Stewart, J. Yosinski, M. A. Gore, R. J. Nelson, H. Lipson, "Automated Identification of Northern Leaf Blight-Infected Maize Plants from Field Imagery Using Deep Learning" Phytopathology, 2017, pp 1426–1432
6. F. N. Iandola, S. Han, M. W. Moskewicz, K. Ashraf, W. J. Dally, and K. Keutzer, "SqueezeNet: AlexNet-level Accuracy with 50x Fewer Parameters and <0.5MB Model Size" Computer Vision and Pattern Recognition, 24 February 2016
7. A. Kamilaris, F. X. Prenafeta-Boldú, "Disaster Monitoring Using Unmanned Aerial Vehicles and Deep Learning" Disaster Management for Resilience and Public Safety Workshop, Proceedings of EnviroInfo, Luxembourg : 2017
8. P. Keane and C. Prior 1991 "Vascular Streak Dieback Of Cacao" Phytopathological Papers, p 33 1–39
9. L. Bin, Y. Zhang, D. He, Y. Li, "Identification of Apple Leaf Diseases Based on Deep Convolutional Neural Networks" Symmetry, 2017
10. M. Afifi and M. S. Brown. "What Else Can Fool Deep Learning? Addressing Color Constancy Errors on Deep Neural Network Performance" International Conference on Computer Vision (ICCV), 2019
11. L. G. Nachtigall, R. M. Araujo, and G. R. Nachtigall, "Classification of Apple Tree Disorders Using Convolutional Neural Networks" In Proceedings of the 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI), San Jose: 6–8 November 2016, pp. 472–476
12. R. Ploetz 2016 "The Impact of Diseases on Cacao Production: A Global Overview" In: Bailey A B and Meinhardt W L, eds. Cacao Diseases: A History of Old Enemies and New Encounters, New York: December 2017
13. R. I. Baihaki and D. R. Sulistyaningrum, "COVID-19 Classification from CT-Scan Images using Convolutional Neural Networks", Proceedings of International Seminar on Machine Learning, Optimization, and Data Science (ISMODE), Jakarta: 29 January 2022, pp. 75–80
14. R. I. Baihaki, Dafik, I. H. Agustin, Z. R. Ridlo, and E. Y. Kurniawati, "The Comparison of Convolutional Neural Networks Architectures on Classification Potato Leaf Disease" in press
15. C. Shorten, T. M. Khoshgoftaar, "A Survey on Image Data Augmentation for Deep Learning" *J Big Data* 6, 60, 2019
16. M. Sibiya, M. A. Sumbwanyambe, "Computational Procedure for The Recognition and Classification of Maize Leaf Diseases Out of Healthy Leaves Using Convolutional Neural Networks" AgriEngineering, 2019, 1, pp 119–131
17. Z. R. Ridlo, I. K. Mahardika, J. Waluyo, R. I. Baihaki, and Dafik, "Design of IOT Based on Nodemcu for Monitoring of Temperature, Soil Moisture, and Relative Humidity as Tools for Precision Agriculture" in press