The Comparison of Convolutional Neural Networks Architectures on Classification Potato Leaf Diseases

Rifki Ilham Baihaki[1], Dafik[1,2], Ika Hesti Agustin[1,3], Zainur Rasyid Ridlo[1,4(✉)], and Elsa Yuli Kurniawati[1]

[1] PUI-PT Combinatorics and Graph, CGANT, University of Jember, Jember, Indonesia
rifkiilham63@gmail.com, {d.dafik,ikahesti.fmipa, zainur.fkip}@unej.ac.id
[2] Departement of Mathematics Education Postgraduate, University of Jember, Jember, Indonesia
[3] Departement of Mathematics, University of Jember, Jember, Indonesia
[4] Department of Science Education, University of Jember, Jember, Indonesia

Abstract. Potato is a plant from the Solanaceae tribe and one of the staple crops for human consumption. Potatoes have several benefits such as being low in fat and having a better carbohydrate content than rice. Behind the relatively easy cultivation of potato plants, there are problems that are often faced by farmers. This problem is the susceptibility of potato plants to disease. An emerging solution is to combine computer vision and deep learning. This research compared four deep learning architectures such as Alexnet, GoogleNet, ResNet-50, and VGG-16. The best model was produced by VGG-16 with a test accuracy of 99.67%.

Keywords: smart agricultural · leaf disease image classification · deep neural networks

1 Introduction

Most of the livelihoods of the people on this earth are farmers. One of the most cultivated plants by farmer is potato [3]. Solanum tuberosum or potato is a plant that is considered to produce more protein and minerals compared to seral [8]. Potatoes originated in South America and were later spread by humans throughout the world. Around 1.5 billion people around the world currently consume potatoes and their processed [5]. Potato is the fourth largest agricultural food crop in the world after maize, wheat, and rice [3]. Behind its benefits, potato plants are susceptible to several diseases that can reduce the quality and yield of agriculture [8].

The agricultural industry faces various problems such as decreased production of apples, tomatoes, potatoes, etc. [6]. To overcome this problem, the proper agricultural precision management is needed. Agricultural precision is a method for managing agricultural production using advanced technology. Agricultural precision produces data which is then used to analyze, predict, or classify the data [2].

I. H. Agustin (Ed.): ICONNSMAL 2022, AISR 177, pp. 125–145, 2023.
https://doi.org/10.2991/978-94-6463-174-6_12

Various techniques and methods are widely used to encourage agricultural production, such as on-farm and off-farm. Research using traditional statistics is conducted to assist statisticians in making decisions. Examples of traditional statistics are logistic regression and discriminant analysis. The assumptions that follow the response variables and predictors are challenging to use. In contrast to statistics, the deep learning approach does not require an understanding of advanced actions and mechanisms [1]. Deep learning can generate predictions to forecast the future data [14].

Advances in remote sensing technology, smartphones, sensors, and data storage on cloud servers using machine learning-based systems can be utilized to make agriculture more intelligent [12]. Machine learning is a subset of artificial intelligence. The goal of machine learning is to make machines learn by recognizing a given object [7]. One method that includes machine learning is an artificial neural network (ANN). This ANN is inspired by biological neural networks that exist in humans. Since it was discovered by McCulloch and Pitts in 1943, this ANN model has undergone various developments such as the emergence of deep learning [4].

Deep learning algorithms contribute to improving computational performance so that researchers are able to obtain large data sets. In recent years, the deep neural network approach has had many positive impacts in agriculture [10]. Advances in computer vision combined with deep learning can be used to diagnose plant diseases quickly and accurately. This advanced technology can increase the precision of agricultural products so that labor costs and time spent monitoring agricultural land are reduced [15]. This technology is also assisted with image processing so as to provide accurate results. Proper diagnosis of potato plant diseases can suppress the spread of this disease [11].

Some diseases that attack potato plants include late blight and late blight. Early blight is caused by a fungal pathogen, namely Alternaria solani and belongs to the phylum Deuteromycota and the order Hyphales. Symptoms of this disease are the appearance of bull's eye spots and concentric rings on some leaf surfaces. Another disease, late blight caused by Phytophthora infectants and is the most vicious disease that attacks potato plants. It has been recorded that there has been a loss of around 6.7 billion US dollars due to this disease in the Americas. Both of these diseases attack potato leaves, then cause them to rot and affect the tubers in the soil [5].

In this study we apply the deep learning method as a feature extractor and classifier. As a comparison we use four architectures namely AlexNet, GoogleNet, ResNet-50, and VGG-16. We categorize potato leaves into three classes, namely healthy class, late blight class, and early blight class.

2 Method

The design of our proposed method is shown in Fig. 1. There are three stages that we carry out, namely (i) Pre-Processing, (ii) Feature Extraction and Pattern Recognition, and (iii) Evaluation.

2.1 Digital Image Processing

The image is defined as a two-dimensional function, $f(x, y)$ where x and y are the spatial coordinates (plane). And the amplitude f in the coordinate pair (x, y) is called

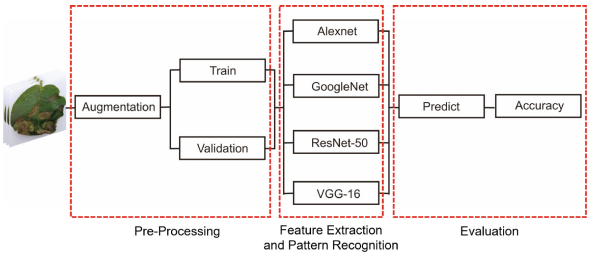

Fig. 1. Proposed Method in This Study.

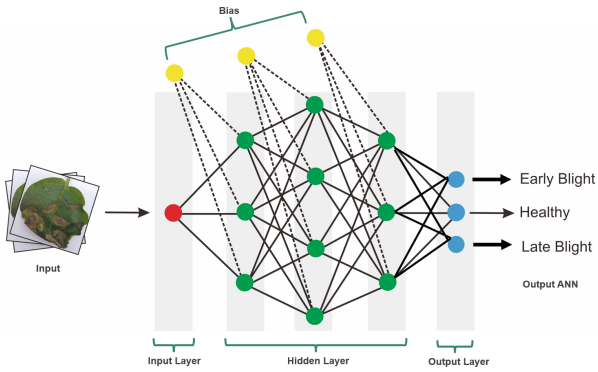

Fig. 2. Types of Layers in Artificial Neural Networks.

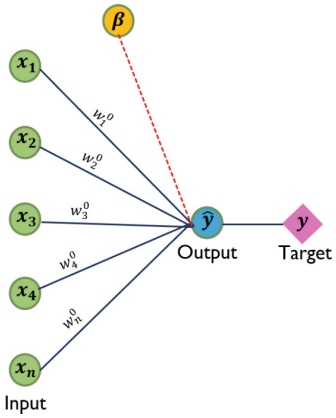

Fig. 3. Simple Artificial Neural Networks.

the intensity or grayscale level of the image at that point. If x, y and intensity values f are all limits, the magnitude is discrete, the image can be said to be a digital image [9].

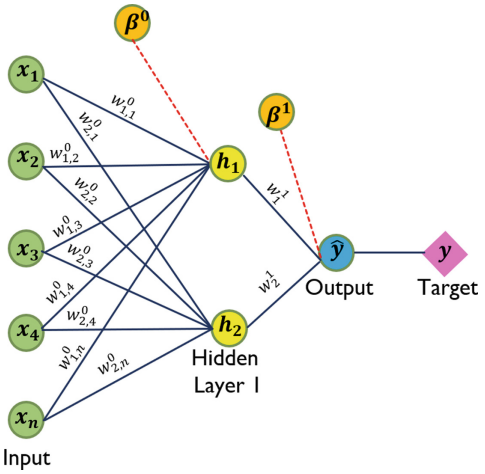

Fig. 4. Illustration of an Artificial Neural Network with One Hidden Layer

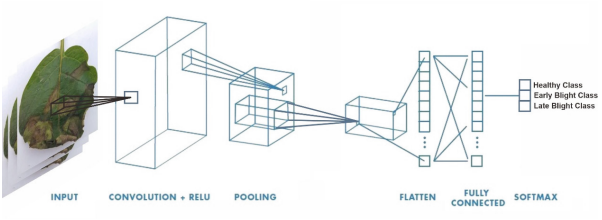

Fig. 5. The Architecture of Convolutional Neural Networks.

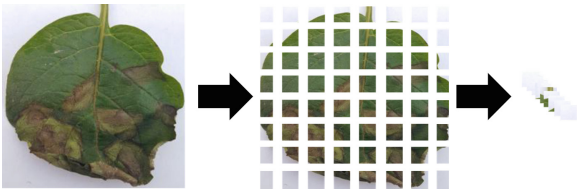

Fig. 6. Splitting Process of Input Image.

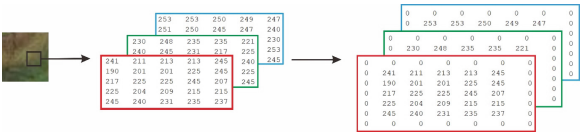

Fig. 7. Added Zero Padding to the Color Channel Matrix.

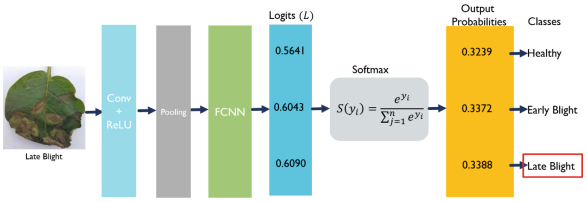

Fig. 8 Illustration of the Softmax Process with Output that Matches the Target.

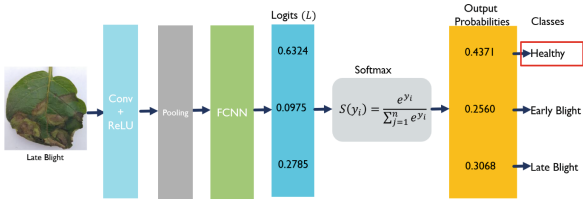

Fig. 9. Illustration of the softmax process with output that does not match the target.

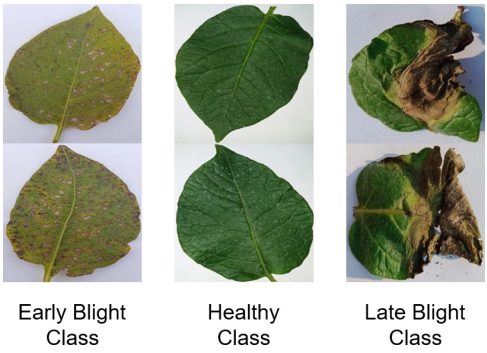

Fig. 10. The Example of Data Set In This Research.

2.2 Artificial Neural Networks

The field of data mining uses a lot of mathematical neural network models (ANN). In ANN there are nodes that represent neurons in the human brain. These nodes are assigned to receive signals from other neurons via synaptic connections. The perceptron is an individual processing element that receives connections from other neurons. The connections between neurons are called weights, and these weights carry encoded electrical information implicitly [4].

In ANN there are three layers that contain one or several nodes, namely the input layer, hidden layer, and output layer. The input layer is the layer that functions to receive information. Furthermore, this information is propagated to other nodes in the hidden layer. The goal is to make the nodes learn the information. The results of this learning then become output on the output layer. Figure 2 illustrates the layers in ANN [7].

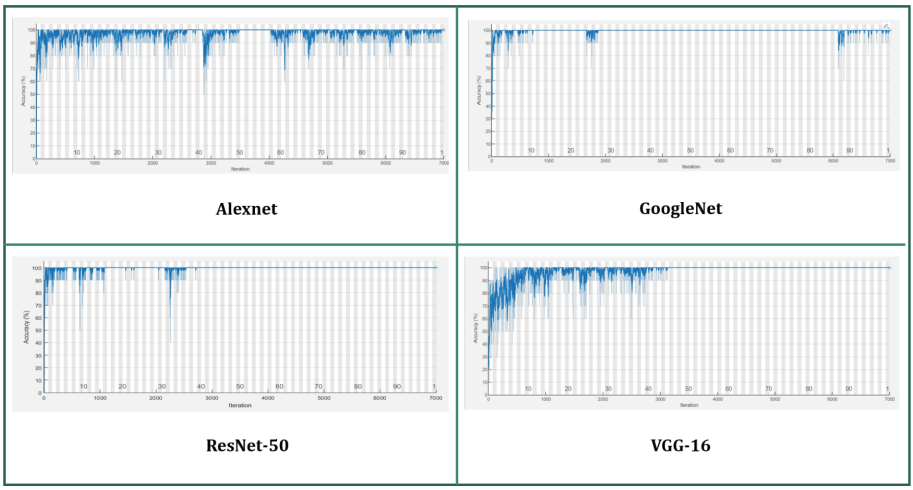

Fig. 11. Comparison of Plot Training on Four CNN Models.

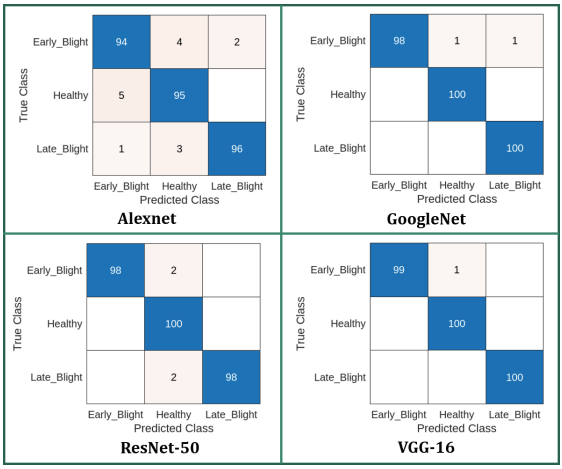

Fig. 12. The Confusion Matrix of Four CNN Models.

Suppose $W = (w_{ij})$ is a matrix that represents the weights, namely the relationship between one neuron and another neuron. Network input to target unit Y_j (with no unit bias j) is a simple dot product of vectors $x = (x_1, x_2, \ldots, x_n)$ where n is the number of inputs dan w_j where j is the number of columns in the weight matrix. Thus, $\widehat{Y} = x \cdot w_j = \sum_{i=1}^{n} x_i w_{ij}$. Figure 3 illustrates a simple neural network. Bias (β) can be included in vector x by adding the component $x_0 = 1$, so the vector x becomes, $x = (1, x_1, x_2, \ldots, x_n)$.

Bias is considered like any other weight, i.e., $w_{0j} = \beta_j$. Network input to the unit Y_j is given as Eq. (1).

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

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 [13]. There are several types of activation functions:

1. Linear activation function $\widehat{Y} = f(x) = ax + b$, When $a = 1, b = 0$, it is an identity.
2. Binary activation function with threshold (θ)

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

3. Binary sigmoid activation function

$\widehat{Y} = f(x) = \frac{1}{1+e^{-\sigma x}}$, dan $f' = \sigma f(x)\left[1 - f(x)\right]$.

4. Bipolar sigmoid activation function

$\widehat{Y} = g(x) = 2f(x) - 1 = \frac{1-e^{-\sigma x}}{1+e^{-\sigma x}}$, dan $g'(x) = \frac{\sigma}{2}\left[1 + g(x)\right]\left[1 - g(x)\right]$.

5. Tangent hyperbolic activation function

$\widehat{Y} = h(x) = \frac{e^{x}-e^{-x}}{e^{x}+e^{-x}}$, dan $h'(x) = [1 + h(x)][1 - h(x)]$.

Backpropagation Algorithm
Step 0. Initialize parameters. Weights (w), bias (β), learning rate (α), and error goal
Step 1. Feedforward stage $h_i = \beta + \sum_{i=0}^{n} x_i \cdot w_{i,j}$ where n is numbers of input.
Step 2. Calculate the output. $y_{in_i} = \beta + \sum_{i=0}^{n} w_{i,j} \cdot h_i$
Step 3. Output activation using log sigmoid. $\hat{y}_m = \frac{1}{1+e^{-y_{in_m}}}$
Step 4. Calculate the output error. $\delta k_i = (t - \hat{y}_i)^2$ $MSE = \frac{\sum_{i=0}^{n}(t-\hat{y}_i)^2}{n}$
Step 5. Calculate the backpropagation error between output layer and hidden layer $\delta j_i^j = \beta \sum_{i=0}^{n} w_{i,j} * \delta k_i$
Step 6. Calculate the backpropagation error between hidden layer and input layer. $\delta j_i^j = \sum_{i=0}^{n} w_{i,j} * \delta k_i$
Step 7. Update weight and bias $W_{new} = W_{old} + \alpha * \delta k_i * h_i$ $\beta_{new} = \beta_{old} + \alpha * \delta j_i + x_i$
Step 8. Checking the termination criteria (epoch reaches the maximum number or $MSE \leq$ error goal)
Step 9. If not, go back to step 1

- **Observation**

Given a neural network with one hidden layer there are two neurons. Figure 4 illustrates this artificial neural network. Suppose $x = \{x_i | i = 1, 2, \ldots, n\}$, where x is input data and n is numbers of input data. Let h dan β respectively be the number of hidden layers and bias. Output ($\hat{y}$) 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 $W^0 = [\, w^0_{1,1}\ w^0_{1,2}\ \cdots\ w^0_{1,n}\ ;\ w^0_{2,1}\ w^0_{2,2}\ \cdots\ w^0_{2,n}\ ]$, $W^1 = [\, w^1_1\ w^1_2\,]$ dan bias β^0, β^1.

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

Next, we get *y_in* as follow

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

So, output ($\hat{y}$) written as follow

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

It concludes the proof. ■

2.3 Convolutional Neural Networks

Convolutional Neural Networks (CNN) are neural networks that have layers with convolution operations in them. The process on CNN begins by breaking the input image into several small overlapping pieces [1]. This cropping of the input image is shown in Fig. 5.

Next is the feature extraction process with the convolution operation. The layer that performs the convolution operation is called the Convolution Layer. Convolution uses a filter of a certain size and performs the matrix multiplication operation. The filter shifts by several pixels starting from the top left corner pixel to the bottom right corner. This pixel shift is called stride. Convolution involves the existing color channel layers in the image such as the red channel layer, the green channel layer, and the blue channel layer. The purpose of this operation is to extract the features in the image. The output of the convolution operation is called a feature map.

Apart from convolution operations, CNN also has other layers such as ReLU and MaxPooling. ReLU (Rectified Linear Unit) is an activation function that is used to generalize the feature map resulting from the convolution operation. Feature map elements with negative values are activated with ReLU and their values change to zero. Meanwhile, feature map elements with a value of zero and positive numbers will not be activated. Furthermore, the MaxPoling process aims to reduce the dimensions of the feature map before being used as input in Fully Connected Neural Networks (FCNN). Suppose there is an image of size $m \times n$, where m represents the number of rows and n represents the number of columns. We build a CNN architecture as shown in Fig. 6.

In the color channel matrix, we provide one row of zero padding. The goal is to maintain the dimensions of the feature map. In addition, we also specify that the convolution stride is one pixel and the filter size is 3×3 with random elements. Illustration of adding zero padding can be seen in Fig. 7.

Convolution operation in the red channel:

$$\begin{aligned} I_{red} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 241 & 211 \\ 0 & 190 & 201 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ -1 & 0 & -1 \end{bmatrix} \\ &= (0 * 1) + (0 * 0) + \cdots + (201 * -1) \\ &= -231 \end{aligned}$$

Convolution operation in the green channel:

$$\begin{aligned} I_{green} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 230 & 248 \\ 0 & 240 & 245 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & -1 \end{bmatrix} \\ &= (0 * 1) + (0 * 0) + \cdots + (245 * -1) \\ &- 263 \end{aligned}$$

Convolution operation in the blue channel:

$$\begin{aligned} I_{blue} &= \begin{bmatrix} 0 & 0 & 0 \\ 0 & 253 & 253 \\ 0 & 251 & 250 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix} \\ &= (0 * -1) + (0 * 0) + \cdots + (250 * -1) \\ &= -250 \end{aligned}$$

Suppose bias (β) = 1, The resulting feature map values are as follows:

$$\begin{aligned} c_1 &= I_{red} + I_{green} + I_{blue} + \beta \\ &= -231 - 263 - 250 + 1 \\ &= -743 \end{aligned}$$

The filter is then shifted one pixel to the right, so that the convolution operation is illustrated as follows:

Convolution operation in the red channel:

$$\begin{aligned} I_{red} &= \begin{bmatrix} 0 & 0 & 0 \\ 241 & 211 & 213 \\ 190 & 201 & 201 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ -1 & 0 & -1 \end{bmatrix} \\ &= (0 * 1) + (0 * 0) + \cdots + (201 * -1) \\ &= -148 \end{aligned}$$

Convolution operation in the green channel:

$$\begin{aligned} I_{green} &= \begin{bmatrix} 0 & 0 & 0 \\ 230 & 248 & 235 \\ 240 & 245 & 231 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & -1 \\ -1 & 0 & -1 \end{bmatrix} \\ &= (0 * 1) + (0 * 0) + \cdots + (231 * -1) \\ &= -458 \end{aligned}$$

Convolution operation in the blue channel:

$$\begin{aligned} I_{blue} &= \begin{bmatrix} 0 & 0 & 0 \\ 253 & 253 & 250 \\ 251 & 250 & 245 \end{bmatrix} * \begin{bmatrix} -1 & 0 & 1 \\ 1 & 0 & 0 \\ 1 & 0 & -1 \end{bmatrix} \\ &= (0 * -1) + (0 * 0) + \cdots + (253 * -1) \\ &= 259 \end{aligned}$$

Then, the resulting feature map values are as follows:

$$\begin{aligned} c_2 &= I_{red} + I_{green} + I_{blue} + \beta \\ &= -148 - 458 + 259 + 1 \\ &= -347 \end{aligned}$$

After going through a series of convolution processes, the following feature map is obtained:

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

$$C = \begin{bmatrix} -743 & -347 & -421 & -388 & 243 \\ -570 & -196 & -146 & 508 & 409 \\ -414 & 17 & 772 & 612 & 521 \\ -256 & 975 & 807 & 670 & 497 \\ 1409 & 1954 & 1833 & 1650 & 683 \end{bmatrix}$$

The next process is to activate the feature map using the Rectified Linear Unit (ReLU). The following is a feature map that has been activated.

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

$$C(\sigma) = \begin{cases} \sigma, & \text{if } \sigma \geq 0 \\ 0, & \text{if } \sigma < 0 \end{cases} = \begin{bmatrix} c_1 & c_2 & \cdots & c_5 \\ c_6 & c_7 & \cdots & c_{10} \\ \vdots & \vdots & \ddots & \vdots \\ c_{21} & c_{22} & \cdots & c_{25} \end{bmatrix}$$

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 243 \\ 0 & 0 & 0 & 508 & 409 \\ 0 & 17 & 772 & 612 & 521 \\ 0 & 975 & 807 & 670 & 497 \\ 1409 & 1954 & 1833 & 1650 & 683 \end{bmatrix}$$

The next process is to reduce the dimensions of the feature map using MaxPooling. The filter used is 3×3 and the stride used is one pixel. The filter moves from the top left corner to the bottom right. The following illustrates the MaxPooling process.

$$m_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 17 & 772 \end{bmatrix}$$

$$\max(m_1) = 772$$

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

$$m_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 508 \\ 17 & 772 & 612 \end{bmatrix}$$

$$\max(m_2) = 772$$

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

$$C = \begin{bmatrix} 772 & 772 & 772 \\ 975 & 975 & 807 \\ 1954 & 1954 & 1833 \end{bmatrix}$$

Next, we transpose the maxpooling result matrix and convert it into a vector as follows.

$$C = [772; 772; 772; 975; 975; 975; 807; 1954; 1954; 1833]$$

Then we normalize the vector C with Eq. 2.

$$x_i = \frac{a + (C_i - \max(C)) \times (b - a)}{\max(C) - \min(C)} \tag{2}$$

where: $a = 0.1$; $b = 0.9$.

So, we obtain the input vector for FCNN as follows

$$x = [0.1; 0.1; 0.1; 0.23; 0.23; 0.12; 0.9; 0.9; 0.81]$$

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 $\alpha = 0.1$, and target (t).

$$W^0 = \begin{bmatrix} 0.1\ 0.2\ 0.3\ 0.1\ 0.2\ 0.3\ 0.1\ 0.2\ 0.3 \\ 0.2\ 0.3\ 0.1\ 0.2\ 0.3\ 0.1\ 0.2\ 0.3\ 0.1 \\ 0.3\ 0.1\ 0.2\ 0.3\ 0.1\ 0.2\ 0.3\ 0.1\ 0.2 \end{bmatrix}$$

$$W^1 = [0.1; 0.2; 0.3]$$

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

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

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

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

$$h^1_{1,1} = \beta^0_1 + x_1 * w^0_{1,1} = 0.1 + 0.1 * 0.1 = 0.1100$$

$$h^1_{1,2} = \beta^0_1 + x_2 * w^0_{1,2} = 0.1 + 0.1 * 0.2 = 0.1200$$

$$\vdots$$

$$h^1_{1,9} = \beta^0_1 + x_9 * w^0_{1,9} = 0.1 + 0.81 * 0.3 = 0.3430$$

$$h^1_1 = \begin{bmatrix} 0.11\ 0.12\ \cdots\ 0.343 \end{bmatrix}$$

Next is feedforward neuron two as follow.

$$h^1_{2,1} = \beta^0_2 + x_1 * w^0_{2,1} = 0.1 + 0.1 * 0.2 = 0.1200$$

$$h^1_{2,2} = \beta^0_2 + x_2 * w^0_{2,2} = 0.1 + 0.1 * 0.3 = 0.1300$$

$$\vdots$$

$$h^1_{2,9} = \beta^0_2 + x_9 * w^0_{2,9} = 0.1 + 0.81 * 0.1 = 0.1810$$

$$h^1_2 = \begin{bmatrix} 0.12\ 0.13\ \cdots 0.181 \end{bmatrix}$$

The last feedforward is in neuron three, the calculate as follow.

$$h^1_{3,1} = \beta^0_3 + x_1 * w^0_{3,1} = 0.1 + 0.1 * 0.3 = 0.1300$$

$$h^1_{3,2} = \beta^0_3 + x_2 * w^0_{3,2} = 0.1 + 0.1 * 0.1 = 0.1100$$

$$\vdots$$

$$h^1_{3,9} = \beta^0_3 + x_9 * w^0_{3,9} = 0.1 + 0.81 * 0.2 = 0.1230$$

$$h^1_3 = \begin{bmatrix} 0.13\ 0.11\ \cdots 0.123 \end{bmatrix}$$

Next is to calculate the output (y)

$$\begin{aligned} yin_1 &= \beta^1_1 + w^1_1 * h^1_{1,1} + w^1_1 * h^1_{1,2} + \cdots + w^1_1 * h^1_{1,9} \\ &= 0.1 + 0.1 * \ 0.1100 + 0.1 * 0.1200 + \cdots + 0.1 * 0.3430 \\ &= 0.2578 \end{aligned}$$

$$\begin{aligned} yin_2 &= \beta^1_2 + w^1_2 * h^1_{2,1} + w^1_2 * h^1_{2,2} + \cdots + w^1_2 * h^1_{2,9} \\ &= 0.1 + 0.2 * 0.1200 + 0.2 * 0.1300 + \cdots + 0.2 * 0.1810 \\ &= 0.4236 \end{aligned}$$

$$\begin{aligned} yin_3 &= \beta^1_3 + w^1_3 * h^1_{3,1} + w^1_3 * h^1_{3,2} + \cdots + w^1_3 * h^1_{3,9} \\ &= 0.1 + 0.3 * 0.1300 + 0.3 * 0.1300 + \cdots + 0.3 * 0.1230 \\ &= 0.4432 \end{aligned}$$

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

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

$$y_1 = 0.5641$$

$$y_2 = 0.6043$$

$$y_3 = 0.6090$$

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

$$\begin{aligned}\delta k_1 &= (t_1 - y_1)^2 \\ &= (1 - 0.5641)^2 \\ &= 0.1900l\end{aligned}$$

$$\begin{aligned}\delta k_2 &= (t_2 - y_2)^2 \\ &= (1 - 0.6043)^2 \\ &= 0.1565\end{aligned}$$

$$\begin{aligned}\delta k_3 &= (t_3 - y_3)^2 \\ &= (1 - 0.6090)^2 \\ &= 0.1529\end{aligned}$$

$$MSE = \frac{((t_1 - y_1) + (t_2 - y_2) + (t_3 - y_3))^2}{3} = 0.4982$$

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

$$\begin{aligned}\delta j_1^1 =& w_{1,1}^0 * \delta k_1 + w_{1,2}^0 * \delta k_1 + w_{1,3}^0 * \delta k_1 + \ldots + w_{1,9}^0 * \Delta k_1 \\ =& 0.3420\end{aligned}$$

$$\begin{aligned}\delta j_2^0 =& \beta_2^0 + w_2^1 * \Delta k_2 \\ =& 0.1 + 0.2 * 0.1565 \\ =& 0.0222\end{aligned}$$

$$\begin{aligned}\delta j_3^0 =& \beta_3^0 + w_3^1 * \Delta k_3 \\ =& 0.1 + 0.3 * 0.1529 \\ =& 0.0331\end{aligned}$$

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

$$\begin{aligned}\delta j_1^1 =& w_{1,1}^0 * \delta k_1 + w_{1,2}^0 * \delta k_1 + w_{1,3}^0 * \delta k_1 + \ldots + w_{1,9}^0 * \delta k_1 \\ =& 0.3420\end{aligned}$$

$$\begin{aligned}\delta j_2^1 =& w_{2,1}^0 * \delta k_2 + w_{2,2}^0 * \delta k_2 + w_{2,3}^0 * \delta k_2 + \ldots + w_{2,9}^0 * \delta k_2 \\ =& 0.2818\end{aligned}$$

$$\delta j_3^1 = w_{3,1}^0 * \delta k_3 + w_{3,2}^0 * \delta k_3 + w_{3,3}^0 * \delta k_3 + \ldots + w_{3,9}^0 * \delta k_3$$

$$=0.2752$$

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

- The weights between input layer and hidden layer

$$\begin{aligned} w\_new^0_{1,1} &= w^0_{1,1} + \alpha * \delta k_1 * h^1_1 \\ &= \begin{bmatrix} 0.1013\ 0.2014\ \cdots\ 0.3041 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} w\_new^0_{2,1} &= w^0_{2,1} + \alpha * \delta k_2 * h^1_2 \\ &= \begin{bmatrix} 0.2016\ 0.3017\ \cdots\ 0.1014 \end{bmatrix} \end{aligned}$$

$$\begin{aligned} w\_new^0_{3,1} &= w^0_{3,1} + \alpha * \delta k_3 * h^1_3 \\ &= \begin{bmatrix} 0.3019\ 0.1016\ \cdots\ 0.1018 \end{bmatrix} \end{aligned}$$

$$W\_new^0 = \begin{bmatrix} w\_new^0_{1,1} \\ w\_new^0_{2,1} \\ w\_new^0_{3,1} \end{bmatrix}$$

$$W\_new^0 = \begin{bmatrix} 0.1013\ 0.2014 \cdots 0.3041 \\ 0.2016\ 0.3017 \cdots 0.1014 \\ 0.3019\ 0.1016 \cdots 0.1018 \end{bmatrix}$$

- The weights between hidden layer and output layer

$$\begin{aligned} w\_new^1_1 &= w^1_1 + \alpha * \delta k_1 \\ &= 0.1355 \end{aligned}$$

$$\begin{aligned} w\_new^1_2 &= w^1_2 + \alpha * \delta k_2 \\ &= 0.2296 \end{aligned}$$

$$\begin{aligned} w\_new^1_3 &= w^1_3 + \alpha * \delta k_3 \\ &= 0.3291 \end{aligned}$$

$$W\_new^1 = \begin{bmatrix} w\_new^1_1 \\ w\_new^1_2 \\ w\_new^1_3 \end{bmatrix} = \begin{bmatrix} 0.1355 \\ 0.2296 \\ 0.3291 \end{bmatrix}$$

- Bias between input layer and hidden layer

$$\beta\_new^0_1 = \beta^0_1 + \alpha \cdot \delta \mathrm{j}_1 = 0.1119$$

$$\beta\_new_2^0 = \beta_1^0 + \alpha \cdot \delta j_2 = 0.1131$$

$$\beta\_new_3^0 = \beta_1^0 + \alpha \cdot \delta j_3 = 0.1146$$

$$\beta_{new}{}^0 = \left[\beta_{new}{}_1^0; \beta_{new}{}_2^0; \beta_{new}{}_3^0\right]$$

$$\beta_{new}{}^0 = [0.1119; 0.1131; 0.1146]$$

- Bias between hidden layer and output layer

$$\beta\_new_1^1 = \beta_1^1 + \alpha \cdot \delta j_1 = 0.1119$$

The process is repeated to the feedforward stage until the number of epochs reaches a maximum. The output value of the neural network is then used as input to the Softmax Layer. In this layer, there will be as many probabilities as the output class we specify. An illustration of the softmax process with output that matches the target shown in Fig. 8. While An illustration of the softmax process with output that matches the target shown in Fig. 9. The output value (logits) is processed using the softmax function. Based on the probability value generated, the system determines the third class as output.

2.4 Confusion Matrix (Metric Performance)

The confusion matrix or commonly called the error matrix is a table that provides comparative information from the classification that has been carried out by the system. The table is presented in Table 1. The confusion matrix can also describe the performance of the classification model on test data.

1. True Positive (TP)

   Is a condition where the model classifies a data as true, and the actual class of the data is true.

2. True Negative (TN)

   Is a condition where the model classifies a data as false, and the actual class of the data is false.

3. False Positive (FP)

   Is a condition where the model classifies a data as true, and the actual class of the data is false.

4. False Negative (FN)

   Is a condition where the model classifies a data as false, and the actual class of the data is true.

Table 1. Table of Confusion Matrix

	True	***False***
True	TP	FP
False	FN	TN

To evaluate the performance of a model, you can use accuracy, precision, and recall. Accuracy shows how much the model gives the correct classification results for the entire data. Accuracy is calculated using Eq. (3).

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \times 100\% \tag{3}$$

Precision shows a comparison of the amount of data that is classified as true with data that is classified as true and data that is classified as false but comes from the true class. Precision can be calculated using Eq. (4).

$$Precision = \frac{TP}{TP + FP} \times 100\% \tag{4}$$

Finally, recall shows the amount of data classified as true with data classified as true and data classified as false but not from the true class. Recall can be calculated using Eq. (5).

$$Recall = \frac{TP}{TP + FN} \times 100\% \tag{5}$$

3 Design and Experiment

The data set we used in this study was taken from the Kaggle Potato Disease Leaf Dataset (PLD) | Kaggle. There are three data classes in this data set, namely early blight class, healthy class, and late blight class. The distribution of the data set we used for network training and testing can be seen in Table 2.

Table 2. The Divided of Data Set

Class	**Data Set**	
	Training	***Testing***
Early Blight	250 Images	100 Images
Healthy	200 Images	100 Images
Late Blight	250 Images	100 Images
Total	**700 Images**	**300 Images**

Table 3. Training and Testing Accuracy

Model CNN	Train Accuracy	Time Train	Test Accuracy
Alexnet	99.85%	3945 s	95%
GoogleNet	100%	6160 s	99.3%
ResNet-50	100%	10588 s	98.67%
VGG-16	100%	28957 s	99.67%

The data set consists of 1000 images, with 700 images used as training and the remaining 300 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. The data set that we use can be seen in Fig. 11.

4 Results and Discussion

4.1 Metrics and Envirenment Setup

In this study, we used Adam optimizer as a performance classification. The number of epochs that we use in this study is 100 epochs. Other parameters at the training stage are the learning rate and batch size. These parameters can be seen in Table 3.

To get optimal CNN model performance we use augmentation data. This technique is used to manipulate data without losing important information in it. The image data that we use is reflected, then a translation is carried out on the x-axis of 30 pixels and -30 pixels on the y-axis.

In this study we used Matlab R2022b software. The computers we use have NVIDIA Cuda Cores 3840 and some hardware: CPU 11th Gen Intel(R) Core (TM) i5-11400H with a speed of 2.70 GHz, GPU NVIDIA GeForce RTX 3060 with 6 GB of memory, and 16 GB of RAM.

4.2 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. We downloaded this model from mathworks.com. Some of these models include Alexnet, GoogleNet, ResNet-50, and VGG-16. Each of these models is identical to the original architecture but without the fully-connected layer.

Alexnet architecture has 25 layers with 5 Convolution Layers, 5 ReLU Layers, and 6 MaxPooling Layers. The GoogleNet architecture has 144 layers with 55 Convolution Layers, 57 ReLU Layers, and 12 MaxPooling Layers. The ResNet-50 Architecture has 177 Layers with 53 Convolution Layers, 49 ReLU Layers, and 1 MaxPooling Layer. The VGG-16 architecture has 41 layers with 13 Convolution Layers, 15 ReLU Layers, and 5 MaxPooling Layers.

When conducting training, each CNN model generates training plots and loss plots. On the training plots we can see the number of epochs and the resulting training accuracy. The higher the training accuracy, the smaller the loss generated by the model.

This training plot can be seen in Fig. 11. Training using the Alexnet, GoogleNet, and ResNet-50 models resulted in increasing and decreasing accuracy. While training using the VGG-16 model the resulting accuracy is 100% stable starting from epoch above 45. Although it produces high training accuracy, the CNN models that we use require a long time to train. The fastest computing time is Alexnet with 1 h of training. While the longest computing time is VGG-16 with about 8 h of training.

4.3 The Evaluation of Testing Stage

In the training process, the models we have trained are stored for use in the testing phase. This stage aims to measure how accurate the model we have trained. Benchmark whether or not this model is determined by the resulting test accuracy value. Based on Table 3, the best CNN model is produced by VGG-16 with a test accuracy of 99.67%. Followed by GoogleNet 99.3%, ResNet-50 98.67%, and Alexnet 95%. Each of these models is then measured its performance with a confusion matrix. The comparison of the confusion matrix in each CNN model is shown in Fig. 12.

Based on Fig. 12, the smallest number of errors is owned by VGG-16. This is why the accuracy of the VGG-16 test reaches 99.67%. The model that generates the most errors is Alexnet. The type of classification that we do is multi-class classification, which means that there are more than two classes that are classified.

So that the precision and recall calculations are different from the binary class. For example, in the Alexnet model, True Positive (TP) for the Early Blight class is 94, True Negative (TN) for the Early Blight class is 191, False Positive (FP) for the Early Blight class is 6, and False Negative (FN) for the Early Blights total 6.

After we know the TP, TN, FP, and FN values of each class, the next step is to calculate accuracy (Eq. 3), precision (Eq. 4), and recall (Eq. 5). In this study, we use accuracy, precision, and recall as benchmarks for the performance of the classification model. Table IV shows the performance of the four classification models.

Based on the table, we can see that the best classification model is produced by VGG-16, with 99.67% accuracy, 99.67% precision, and 99.67% 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 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

The agricultural sector is the livelihood with the largest number of workers in Indonesia. The production of agricultural products is also influenced by several factors, one of which is agricultural precision. Just like humans, plants can also get sick. So, it is necessary to create a system model that is able to classify the types of plant diseases automatically. In

our research we apply deep learning to classify the types of diseases that exist on potato leaves. As a comparison, we used four CNN models that are popular among researchers to see which model is more effective in our problem. Based on the results we obtained, it can be concluded that the best model was produced by VGG-16 with an accuracy of 99.67%. We can then use the VGG-16 model to classify potato leaf diseases from the data taken directly. Of course, to make machines able to recognize types of diseases like humans, learning and testing is needed. The learning phase aims to train the machine, while the testing phase aims to measure the performance of the training model.

Acknowledgment. We gratefully acknowledge the assistance and support of PUI-PT Combinatorics and Graph, CGANT, University of Jember 2023.

References

1. B. T. W. Putra, R. Amirudin, B. Marhaenanto, "The Evaluation of Deep Learning Using Convolutional Neural Network (CNN) Approach for Identifying Arabica and Coffe Plants", *Journal of Biosystems Engineering*, 2022.
2. C. Wang, B. Liu, L. Liu, Y. Zhu, J. Hou, P. Liu, and X. Li, " A Review of Deep Learning used in The Hyperspectral Image Analysis for Agriculutre", *Artificial Intelligence Review*, 2021.
3. D. Tiwari, M. Ashish, N. Gangwar, A. Sharma, S. Patel, and S. Bhardwaj, "Potato Leaf Disease Detection using Deep Learning", *Proceedings of the International Conference on Intelligent Computing and Control Systems*, vol. 1, pp. 461-466, 2020.
4. K. Golhani, S. K. Balsundram, G. Vadamalai, and B. Pradhani, "A Review of Neural Networks in Plant Disease Detection using Hyperspectral Data", *Information Processing in Agriculture*, vol. 5, pp. 354-371, 2018.
5. K. K. Chakraborty, R. Mukherje, C. Chakraborty, and K. Bora. "Automate Recognition of Optical Image Based Potato Leaf Blight Disease Using Deep Learning", *Physiological and Molecular Plant Pathology*, vol. 1, pp. 1-10, 2022.
6. M. H. Saleem, J. Potgieter, and K. M. Arif, "Automation in Agriculture by Machine and Deep Learning Techniques: A Review of Recent Developments", *Precision Agriculture*, 2021
7. M. Pathan, N. Patel, H. Yagnik, and M. Shah. "Artificial Cognition for Application in Smart Agriculture: A Comprehensive Review", *Artificial Intellegence in Agriculture.* vol. 4, pp. 81-95, 2020.
8. N. E. M. Khalifa, N. H. M. Taha, L. M. A. El-Maged, and A. E. Hassanaein, "Artificial Intellegence in Potato Leaf Disease Classification: A Deep Learning Approach", *Machine Learning and Big Data Analytics Paradigms*, vol. 77, pp. 63-79, 2021.
9. R. C. Gonzales and R. E. Woods, "*Digital Image Processing Fourth Edition*". England: Pearson Education, 2018.
10. S. Coulibaly, B. Kamsu-Foguem, D. Kamissoko, D. Traore, "Deep Learning for Precision Agriculture: A Bibliometric Analysis", *Intelligent Systems with Applications*, vol. 16, pp. 1 – 18, 2022.
11. T. A. Shaikh, T. Rasool, and F. R. Lone, "Towards Leveraging the Role of Machine Leaning and Artificial Intelligence in Precious Agriculture and Smart Farming", *Computers and Electronics in Agriculture*, vol. 198, pp. 1 – 29, 2021.
12. V. G. Dhanya, A. Subeesh, N. L. Kushwaha, D. K. Vishwakarma, T. N. Kumar, G. Ritika, A. N. Singh, "Deep Learning based Computer Vision Approaches for Smart Agricultural Applications", *Artificial Intelligence in Agriculture*, vol. 6, pp. 211 – 229, 2022.

13. 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.
14. Dafik, Q. A. A'yun, R. I. Baihaki, A. C. Prihandoko, A. I. Kristiana, F. G. Febrinanto, and K. A. Santosa, "The Spatial Temporal Graph Neural Networks and Rainbow Antimagic Coloring for Time Series Forecasting on Flood Flow Anomaly" in press.
15. 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 Precission Agriculture" in press.