Classification of Disease in Rice Plant Leaves Using the Method *Convolutional Neural Networks*

Laila Badriyatuz Zahro[1], Dafik[2,3], Ika Hesti Agustin[1](✉), and Zainur Rasyid Ridlo[2,4]

[1] Department of Mathematics, University of Jember, Jember, Indonesia
ikahesti.fmipa@unej.ac.id
[2] PUI-PT Combinatroics and Graph, CGANT, University of Jember, Jember, Indonesia
[3] Department of Mathematics Education Postgraduate, University of Jember, Jember, Indonesia
[4] Department of Science Education, University of Jember, Jember, Indonesia

Abstract. Rice plant disease is one of the factors causing high losses due to crop failure. Plant-disturbing organisms often attack rice plants, especially on the leaves. This can damage rice plants and cause crop failure. Manual diagnostic activities on rice plant leaves will help identify and classify the types of diseases suffered by rice plant leaves. This study aims to be able to detect diseases that occur in the leaves of rice plants using the Convolutional Neural Network (CNN) method. Convolutional Neural Network is one method that is quite effective for image classification. Image will go through the Pre-Processing, Feature Extraction and Evaluation processes. The dataset used is RiceLeafs Diseases from kaggle with a total of 3000 samples of rice leaf images, 2100 images for training and 300 images for testing. In our research we used 3 different epoch numbers to find the value that produces the highest accuracy. Based on the research, it was found that 75 epochs had the highest accuracy value namely 85.67%.

Keywords: rice plant diseases · convolutional neural networks · image classification

1 Introduction

Rice is a plant commodity that has been cultivated by farmers, especially in Indonesia, for centuries. Plants with high economic value will be needed forever because rice is a rice-producing crop for food consumption and nutritional needs for all mankind. However, Plant Pest Organisms (OPT) often disturb rice plants, especially on the leaves, this can cause crop failure resulting in huge losses for farmers [1]. The cultivation of rice plants is inseparable from the threat of pests and diseases that often attack plants if the control is not appropriate can reduce the productivity of rice plants [10]. Therefore, a manual diagnosis is needed to identify the types of pests and diseases on the leaves of rice plants so that farmers can identify and apply appropriate controls. By controlling pests and diseases, the cultivation goals will be achieved [2].

The current development of science has encouraged the discovery of ways to detect diseases in plants automatically by using a computer. Disease detection using a computer is quite recommended because computer-generated detection is considered quite accurate. Several studies on the detection of plant pests and diseases have been carried out and some of them use the Deep Learning method [3].

Utilization of Machine Learning (ML) technology is an effective tool in increasing the effectiveness of control management systems in agriculture [4]. Machine learning is part of artificial intelligence. The goal of machine learning is to create machine learning by recognizing given object [7]. One method that includes machine learning is artificial neural network (ANN) [5]. Deep Learning is part of Machine Learning which uses the Neural Network method to solve a problem. *Convolutional Neural Network* (CNN) is one of the Deep Learning algorithms that is often used to solve image classification problems [8]. CNN is known to produce a significant level of accuracy because this method has a high network depth and is able to study the features contained in complex images. Therefore, in this research we carried out the study on disease classification in rice plant leaves *Convolutional Neural Network* (CNN).

In this study we will classify the disease in rice plant leaves. The method used is Deep Learning using a *Convolutional Neural Network* (CNN). The input data used is the image of diseased rice plant leaves, then the image will be classified to determine whether there is disease on the rice plant leaves. The plants we used were rice leaves plants with 3 types of disease, namely Brownspot, Hispa, and Leaf Blast, bringing the total to 3 classes. Our classification is only done to detect one disease from one leaf image. The image data taken has a fairly clear lighting.

A. Leaf Disease of Rice Plants

1) *Brownspot*

Disease caused by the fungus *Cochliobolus miyabeanus*. Symptoms start with round or oval brown spots with a yellow halo. On susceptible varieties, the lesions can be 5–14 mm long and can cause wilting of the leaves. This causes incomplete grain filling which can reduce grain quality. An example of an image can be seen in Fig. 1 [1].

2) *Hispa*

Disease caused by adult insects and rice hispa larvae (*Dicladispa Armigera*) by eroding the upper surface of the leaf blade and leaving only the lower epidermis. Adult's insects feed on the outer epidermis causing white patches to appear. The larvae feed on the green tissue between the leaf epidermis which also causes white spots. Affected leaves dry up and from a distance the badly damaged land looks like i on fire. An example of an image can be seen in Fig. 2 [1].

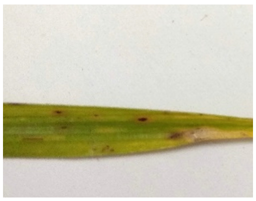

Fig. 1. Brownspot

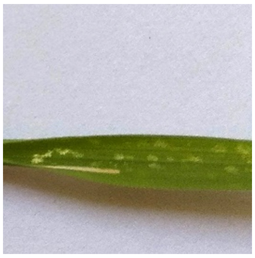

Fig. 2. Hispa

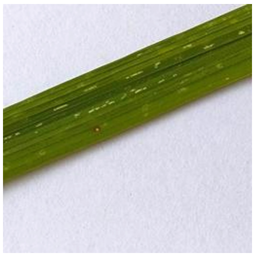

Fig. 3. Leafblast

3) Leaf Blast

Disease caused by the fungus *Pyricularia Grisea.* At the growth stage of seedlings and rice, this fungus infects the leaves and causes disease symptoms in the form of brown rhombuses which can result in death of the rice plants. An example of an image can be seen in Fig. 3 [9].

2 Method

The design method we propose can be seen in Fig. 4. There are three stages that we carry out, namely (i) Pre-Processing, (ii) Feature Extraction, (iii) Evaluation [5].

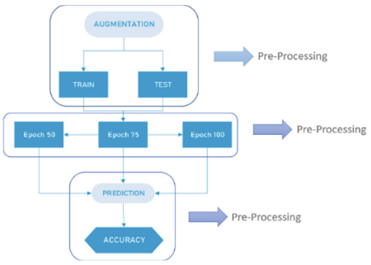

Fig. 4. Proposed Method in This Study

A. *Digital Image Processing*

The image is defined as a two-dimensional function, $f(x, y)$ where and are the spatial coordinates (plane), and the amplitude f in the coordinate pair (x, y) is called 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 [6].

2.1 Backpropagation Neural Network

Suppose $W = (w_{ij})$ is a matrix that represents the weights, namely the relationship between one neuron and another neuron. The network input to the target unit Y_j (with no bias to unit j) is the simple dot product of the vector $x = (x_1, x_2, \ldots, x_n)$ where n is the number of inputs and w_j where j is the number of columns in the weight matrix. So that $\widehat{Y} = x.w_j = \sum_{i=1}^{n} x_i w_{ij}$ Fig. 5 illustrates a simple neural network. The bias (β) can be included in vector x by adding vector components $x_0 = 1$, so that vector x becomes $x = (1, x_1, x_2, \ldots, x_n)$.

Bias is considered like other weights, namely $w_{0j} = \beta_j$. The network input to unit Y_j is given as Eq. (1) below

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

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. There are several types of activation functions.

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

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

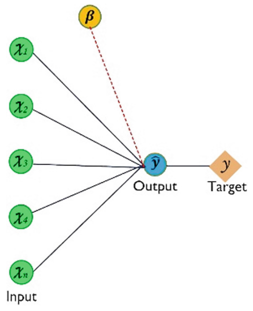

Fig. 5. Simple Artificial Neural Network

3. Sigmoid binary activation function

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

4. Sigmoid bipolar activation function

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

5. Hyperbolic tangent activation function

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

```
Backpropagation Algorithm
Step 0. Initialize the weights $\boldsymbol{w}$, $\boldsymbol{\beta}$, $\boldsymbol{\alpha}$, error_goal.
Step 1. Feedward Stage
    $\boldsymbol{h_i = \beta + \sum_{i=0}^{n} x_i \cdot w_{i,j}}$ where $\boldsymbol{n}$ is the number of input.
Step 2. Calculating Output.
    $\boldsymbol{y_{in_i} = \beta + \sum_{i=0}^{n} w_{i,j} \cdot h_i}$
Step 3. Output activation using sigmoid log.
    $\boldsymbol{\hat{y}_m = \frac{1}{1+e^{-y_{in_m}}}}$
Step 4. Calculate the error output.
    $\boldsymbol{\delta k_i = \left(t - \hat{y}_i\right)^2}$
    $\boldsymbol{MSE = \frac{\sum_{i=0}^{n}\left(t - \hat{y}_i\right)^2}{n}}$
Step 5. Calculating Backpropagation Error between output layer and hidden layer.
    $\boldsymbol{\delta j_i^j = \beta \sum_{i=0}^{n} w_{i,j} * \delta k_i}$
Step 6. Update weights and bias.
    $\boldsymbol{W_{new} = W_{old} + \alpha * \delta k_i * h_i}$
    $\boldsymbol{\beta_{new} = \beta_{old} + \alpha * \delta j_i + x_i}$
Step 7. Checking the termination criteria (epoch reaches the maximum number or $\boldsymbol{MSE} \leq$ error_goal)
Step 8. If not, go back to step 1.
```

- **Observation 1**

Given a neural network with one hidden layer there are two neurons. Figure 6 illustrates this artificial neural network. For example $x = \{x_i | i = 1, 2, \ldots, n\}$, 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 bias 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 bobot.

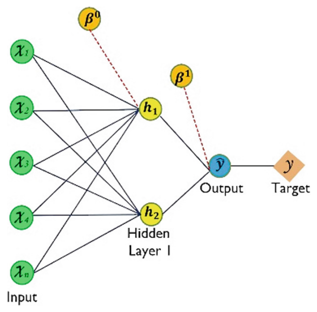

Fig. 6. Artificial Neural Network with 1 Hidden layer

$W^0 = [\, w^0_{1,1}\ w^0_{1,2}\ \cdots\ w^0_{1,n}\ ;\ w^0_{2,1}\ w^0_{2,2}\ \cdots\ w^0_{2,n}\ ] W^1 = [\, w^1_1\ w^1_2\,]$ and bias β^0, β^1.

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

Next we obtain *y_in* as follows

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

$$= \beta^1 + W^1\left(\beta^0 + W^0 x\right)$$

So that the resulting output $(\hat{y})$ is as follows

$$\hat{y} = \frac{1}{1 + e^{-y_{in}}}$$

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

That 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 7 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 $\hat{y} = \frac{1}{1+e^{-(\beta_1 + \sum_{i=1}^{2} w_i h_i)}}$ where $[h_1; h_2] = \beta^0 + W^0 x$.

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$W^0 =$

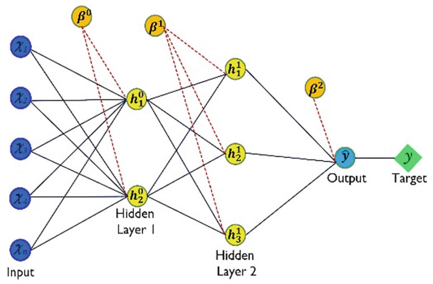

Fig. 7. Artificial Neural Network with 2 Hidden Layer

$[w^0_{1,1}\ w^0_{1,2}\ \ldots\ w^0_{1,n};\ w^0_{2,1}\ \ w^0_{2,2}\ \ldots\ w^0_{2,n}\ ],W^1 = [w^1_{1,1}\ w^1_{1,2};\ w^1_{2,1}\ w^1_{2,2};\ w^1_{3,1}\ w^1_{3,2}]$, $W^2 = [w^2_1;\ w^2_2;\ w^2_3]$ and bias$\beta_0, \beta_1, \beta_2$. The output from hidden layer 1 is as follows.

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

The output of hidden layer 2 is as follows

$$\begin{bmatrix} h^2_1 \\ h^2_2 \\ h^2_3 \end{bmatrix} = \begin{bmatrix} \beta^1_1 \\ \beta^1_2 \\ \beta^1_3 \end{bmatrix} + \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,n} \\ w^1_{2,1} & w^1_{2,1} & \cdots & w^1_{2,n} \\ w^1_{3,1} & w^1_{3,2} & \ldots & w^1_{3,2} \end{bmatrix} \begin{bmatrix} h^1_1 \\ h^1_2 \end{bmatrix}$$
$$= \begin{bmatrix} \beta^1_1 \\ \beta^1_2 \\ \beta^1_3 \end{bmatrix} + \begin{bmatrix} w^1_{1,1} & w^1_{1,2} & \cdots & w^1_{1,n} \\ w^1_{2,1} & w^1_{2,1} & \cdots & w^1_{2,n} \\ w^1_{3,1} & w^1_{3,2} & \ldots & w^1_{3,2} \end{bmatrix} \cdot (\beta^0 + W^0 x)$$
$$= \beta^1 + W^1(\beta^0 + W^0 x)$$

Next we obtain *y_in* as follows

$$y_{in} = \beta^2 + \begin{bmatrix} w^2_1 \\ w^2_2 \\ w^2_3 \end{bmatrix} \begin{bmatrix} h^2_1 \\ h^2_2 \\ h^2_3 \end{bmatrix} = \beta^2 + \begin{bmatrix} w^2_1 \\ w^2_2 \\ w^2_3 \end{bmatrix} \cdot \left(\beta^1 + W^1\left(\beta^0 + W^0 x\right)\right)$$
$$= \beta^2 + W^2\left(\beta^1 + W^1\left(\beta^0 + W^0 x\right)\right)$$

So that the resulting output ($\hat{y}$) is as follows

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

That concludes the proof.

- **Illustration**

Suppose there is an image of size $m \times n$, where m represents the number of rows and n represents the number of columns. The CNN image is divided into several parts with

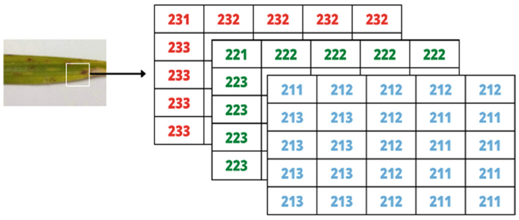

Fig. 8. Input Image

each part we perform the convolution operation. In each fragment of the image there are three constituent color channels, namely the red, green and blue color channels. The three colors contain a matrix like in the Fig. 8.

In the color channel matrix, we provide one row of zero padding. The goal is to maintain the dimensions of the feature map. Besides that, 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 the Fig. 9.

Convolution operation in the red channel:

$$I_{red} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 231 & 232 \\ 0 & 233 & 233 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ -1 & 0 & -1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 0) + \cdots + (233 * -1)$$

$$= -232$$

Convolution operation in the green channel:

$$I_{green} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 221 & 222 \\ 0 & 223 & 223 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 0) + \cdots + (223 * -1)$$

$$= 443$$

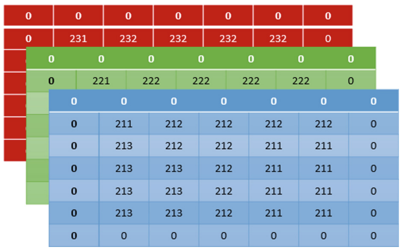

Fig. 9. Adding Zero Padding to the Matrix

Convolution operation in the blue channel:

$$I_{blue} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 211 & 212 \\ 0 & 213 & 213 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 1) + \cdots + (213 * -1)$$

$$= 212$$

Suppose bias $(\beta) = 1$, the resulting feature map values are as follows:

$$C_1 = I_{red} + I_{green} + I_{blue} + \beta$$

$$= -232 + 443 + 212 + 1$$

$$= 424$$

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:

$$I_{red} = \begin{bmatrix} 0 & 0 & 0 \\ 231 & 232 & 232 \\ 233 & 233 & 232 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 1 \\ 1 & -1 & 1 \\ -1 & 0 & -1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 0) + \cdots + (232 * -1)$$

$$= -234$$

Convolution operation in the green channel:

$$I_{green} = \begin{bmatrix} 0 & 0 & 0 \\ 221 & 222 & 222 \\ 223 & 223 & 222 \end{bmatrix} * \begin{bmatrix} 1 & 0 & 0 \\ -1 & 1 & 1 \\ 0 & 1 & -1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 0) + \cdots + (222 * -1)$$

$$= 224$$

Convolution operation in the blue channel:

$$I_{blue} = \begin{bmatrix} 0 & 0 & 0 \\ 211 & 212 & 212 \\ 213 & 213 & 212 \end{bmatrix} * \begin{bmatrix} 1 & 1 & 0 \\ 1 & 0 & 1 \\ 1 & 1 & -1 \end{bmatrix}$$

$$= (0 * 1) + (0 * 1) + \cdots + (212 * -1)$$

$$= 637$$

Then, the resulting feature map values are as follows:

$$C_2 = I_{red} + I_{green} + I_{blue} + \beta$$

$$= -234 + 224 + 637 + 1 = 628$$

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

$$C = \begin{bmatrix} C_1 & C_2 & \ldots & C_5 \\ C_6 & C_7 & \ldots & C_{10} \\ \vdots & \vdots & \ddots & \vdots \\ C_{21} & C_{22} & \ldots & C_{25} \end{bmatrix}$$

$$= \begin{bmatrix} 424 & 628 & 630 & 628 & 625 \\ 870 & 1737 & 1738 & 1735 & 1502 \\ 873 & 1744 & 1278 & 1733 & 1498 \\ 873 & 1744 & 1740 & 1733 & 1498 \\ 1106 & 1994 & 1989 & 1984 & 1086 \end{bmatrix}$$

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.

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

$$C = \begin{bmatrix} 0 & 0 & 0 & 0 & 625 \\ 0 & 0 & 0 & 1735 & 1502 \\ 0 & 1744 & 1278 & 1733 & 1498 \\ 0 & 1744 & 1740 & 1733 & 1498 \\ 1106 & 1994 & 1989 & 1984 & 1086 \end{bmatrix}$$

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.

$$m_1 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 1744 & 1278 \end{bmatrix}$$

$$\max(m_1) = -1278$$

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

$$m_2 = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1735 \\ 1744 & 1278 & 1733 \end{bmatrix}$$

$$\max(m_2) = -1096$$

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

$$C = \begin{bmatrix} -1275 & -1096 & -638 \\ -41 & 1327 & 2117 \\ 1918 & 3254 & 2321 \end{bmatrix}$$

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

$$C = [-1275; -1096; -638; -41; 1327; 2117;$$

$$1918; 3254; 2321]$$

Then we normalize the vector C with the following formula

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

where$a = 0.1; b = 0.9$.

So, we obtain the input vector for FCNN as follows.

$$x = [0.15; 0.25; 0.78; 0.83; 0.37; 0.92; 0.61; 0.13; 0.43]$$

These vectors then become input to Fully-Connected Neural Networks (FCNN). Some of the parameters we used in FCNN are weights $(W^0 and W^1)$, bias $(\beta^0 and \beta^1)$ learning rate $\alpha = 0.1$ and target (t).

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

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

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

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

$$t = [111]$$

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

$$h^1_{1,1} = \beta^0_1 + x_1 * w^0_{1,1} = 0.1 + 0.15 * 0.1 = 0.1150$$

$$h^1_{1,2} = \beta^0_1 + x_2 * w^0_{1,2} = 0.1 + 0.25 * 0.2 = 0.1500$$

$$\vdots$$

$$h^1_{1,9} = \beta^0_1 + x_9 * w^0_{1,9} = 0.1 + 0.43 * 0.3 = 0.2290$$

$$h^1_1 = [0.1150; 0.1500; 0.3340; 0.1830; 0.1740;$$

$$0.3760; 0.1610; 0.1260; 0.2290]$$

Next is feedforward neuron two as follows.

$$h^1_{2,1} = \beta^0_2 + x_1 * w^0_{2,1} = 0.1 + 0.15 * 0.3 = 0.1450$$

$$h^1_{2,2} = \beta^0_2 + x_2 * w^0_{2,2} = 0.1 + 0.25 * 0.2 = 0.1500$$

$$\vdots$$

$$h^1_{2,9} = \beta^0_2 + x_9 * w^0_{2,9} = 0.1 + 0.43 * 0.1 = 0.1430$$

$$h^1_2 = [0.1450; 0.1500; 0.1780; 0.3490; 0.1740;$$

$$0.1920; 0.2830; 0.1260; 0.1430]$$

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

$$h^1_{3,1} = \beta^0_3 + x_1 * w^0_{3,1} = 0.1 + 0.15 * 0.2 = 0.1300$$

$$h^1_{3,2} = \beta^0_3 + x_2 * w^0_{3,1} = 0.1 + 0.25 * 0.3 = 0.1750$$

$$\vdots$$

$$h^1_{3,9} = \beta^0_3 + x_9 * w^0_{3,9} = 0.1 + 0.43 * 0.1 = 0.2110$$

$$h^1_3 = [0.1300; 0.1750; 0.1780; 0.2660; 0.2110;$$

$$0.2110; 0.2110; 0.2110; 0.2110]$$

Next is to calculate the output (y)

$$yin_1 = \beta_1^1 + w_1^1 * h_{1,1}^1 + w_1^1 * h_{1,2}^1 + \cdots + w_1^1 * h_{1,9}^1$$

$$= 0.1 + 0.1 * 0.1150 + 0.1 * 0.1500 + \cdots + 0.1 * 0.2290$$

$$= 0.2848$$

$$yin_2 = \beta_2^1 + w_2^1 * h_{2,1}^1 + w_2^1 * h_{2,2}^1 + \cdots + w_2^1 * h_{2,9}^1$$

$$= 0.1 + 0.2 * 0.1450 + 0.2 * 0.1500 + \cdots + 0.2 * 0.1430$$

$$= 0.4480$$

$$yin_3 = \beta_3^1 + w_3^1 * h_{3,1}^1 + w_3^1 * h_{3,2}^1 + \cdots + w_3^1 * h_{3,9}^1$$

$$= 0.1 + 0.3 * 0.1300 + 0.3 * 0.1750 + \cdots + 0.3 * 0.2110$$

$$= 0.6412$$

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

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

$$y_1 = 0.5707$$

$$y_2 = 0.6102$$

$$y_3 = 0.6550$$

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

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

$$= (1 - 0.5707)^2 = 0.1843$$

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

$$= (1 - 0.6102)^2 = 0.1520$$

$$\delta k_3 = (t_3 - y_3)^2$$

$$= (1 - 0.6550)^2 = 0.1190$$

$$MSE = \frac{((t_1 - y_1) + (t_2 - y_2) + (t_3 - y_3))^2}{3} = 0.4517$$

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

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

$$= 0.1 + 0.1 * 0.1843 = 0.1184$$

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

$$= 0.1 + 0.2 * 0.1520 = 0.1304$$

$$\delta j_3^0 = \beta_3^0 + w_3^1 * \delta k_3$$

$$= 0.1 + 0.3 * 0.1190 = 0.1357$$

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

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

$$= 0.3317$$

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

$$= 0.2736$$

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

$$= 0.2142$$

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

- The weights between input layer and hidden layer

$$w\_new_{1,1}^0 = w_{1,1}^0 + \alpha * \delta k_1 * h_1^1 = [0.1014; 0.2018; 0.3040; 0.1022; 0.2021;$$

$$0.3045; 0.1019; 0.2015; 0.3027]$$

$$w_{new}{}^{0}_{2,1} = w^{0}_{2,1} + \alpha * \delta k_2 * h^{1}_{2} = [0.3019; 0.2020; 0.1023; 0.3046; 0.2023; 0.1025$$

$$0.3027; 0.2016; 0.1019]$$

$$w_{new}{}^{0}_{3,1} = w^{0}_{3,1} + \alpha * \delta k_3 * h^{1}_{3} = [0.2018; 0.3024; 0.1024; 0.2036; 0.3029; 0.3029$$

$$0.3029; 0.3029; 0.3029]$$

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

$$W\_new^{0} = \begin{bmatrix} 0.1014\ 0.2018\ 0.3040 \cdots 0.3027 \\ 0.3019\ 0.2020\ 0.1023 \cdots 0.1019 \\ 0.2018\ 0.3024\ 0.1024 \cdots 0.3029 \end{bmatrix}$$

- The weights between hidden layer and output layer

$$w\_new^{1}_{1} = w^{1}_{1} + \alpha * \delta k_1$$

$$= 0.1345$$

$$w\_new^{1}_{2} = w^{1}_{2} + \alpha * \delta k_2$$

$$= 0.2291$$

$$w\_new^{1}_{3} = w^{1}_{3} + \alpha * \delta k_3$$

$$= 0.3254$$

$$W\_new^{1} = \begin{bmatrix} w\_new^{1}_{1} \\ w\_new^{1}_{2} \\ w\_new^{1}_{3} \end{bmatrix} = \begin{bmatrix} 0.1354 \\ 0.2291 \\ 0.3254 \end{bmatrix}$$

- Bias between input layer and hidden layer

$$\beta\_new^{0}_{1} = \beta^{0}_{1} + \alpha * \delta j_1$$

$$= 0.1118$$

$$\beta\_new^{0}_{2} = \beta^{0}_{2} + \alpha * \delta j_2$$

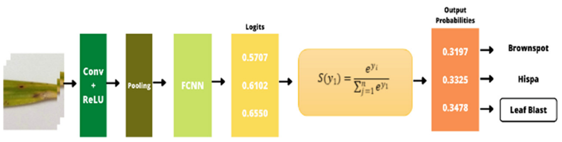

Fig. 10. Softmax Process with Output that Matches the Target

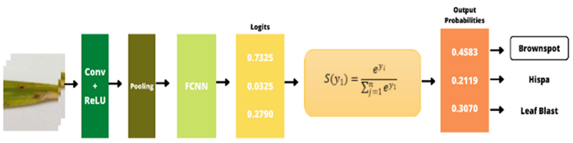

Fig. 11. Softmax Process with Output that Does Not Matches the Target

$$= 0.1130$$

$$\beta\_new_3^0 = \beta_3^0 + \alpha * \delta j_3$$

$$= 0.1136$$

$$\beta\_new^0 = \begin{bmatrix} \beta\_new_1^0 \\ \beta\_new_2^0 \\ \beta\_new_3^0 \end{bmatrix} = \begin{bmatrix} 0.1118 \\ 0.1130 \\ 0.1136 \end{bmatrix}$$

- Bias between hidden layer and output layer

$$\beta\_new_1^1 = \beta_1^1 + \alpha * \delta j_1 = 0.1118$$

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 is shown in Fig. 10. The output value (logits) is processed using the softmax function. Based on the probability value generated, the system determines the third class as output (Fig. 11).

2.2 Confusion Matrix

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.

True Positive (TP)

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

True Negative (TN)

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

False Positive (FP)

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

False Negative (FN)

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

To evaluate the performance of a model, you can use accuracy, precision, and recall. Accuration shows how much the model gives the correct classification results for the entire data. Accuration is calculated using Eq. (1).

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

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. (2).

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

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. (3).

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

Table 1. Confusion Matrix

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

Table 2. The Divide of Dataset

Class	**Dataset**	
	Training	***Testing***
Brownspot	700 Images	300 Images
Hispa	700 Images	300 Images
Leafblast	700 Images	300 Images
Total	**2100 Images**	**900 Images**

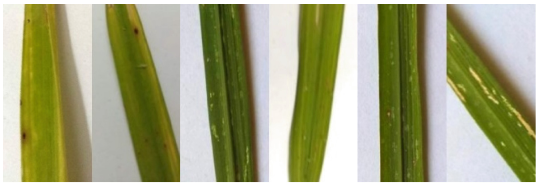

Fig. 12. The Example of Data Set in This Research

3 Experiment

The data set we used in this study was taken from the Kaggle Rice Disease Leaf Dataset | Kaggle. There are three classes of data, namely using epoch 50, epoch 75, and epoch 100. The distribution of the data sets used for network training and testing can be seen in Table 2.

This dataset consists of 3000 images, which 2100 images are used as training and 900 images are used as tests. The training stage aims to train the neural network to recognize the input image provided. The output of network training is training accuracy and training models. In the testing phase, this model is used to recognize new data. The dataset we use can be seen in Fig. 12.

4 Results and Discussion

In this study we used a different number of epochs, namely 50 epochs, 75 epochs, 100 epochs. Accuracy values and accuracy time can be seen in Table 3.

4.1 Training Evaluation Stage

In this study we used basic CNN, commonly used in image data. The image will be trained using different epoch values to find high accuracy results. When doing training, we can see the number of epoch and the resulting training accuracy. The higher the training accuracy, the smaller the resulting loss. The training plot can be seen in Fig. 13.

Table 3. Training and Testing Accuracy

Number of Epochs	Train Accuracy	Time Train	Test Accuracy
Epoch 50	81%	6550 s	75%
Epoch 75	86%	9825 s	85,7%
Epoch 100	85%	13100 s	82,3%

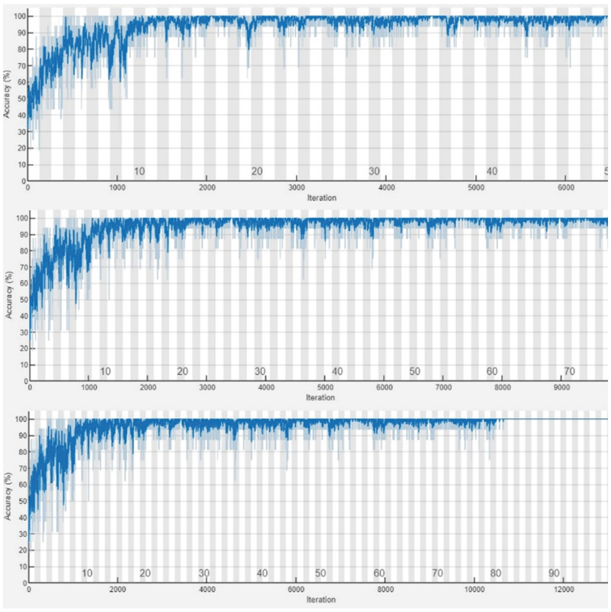

Fig. 13. Comparison of Plot Training

Training using 50 epochs, 75 epochs and 100 epochs resulted in increasing and decreasing accuracy. But training using 75 epochs produces a more stable accuracy. Even though it produces a high accuracy value, the number of epochs determines the long time to do the training. The fastest time is 50 epochs with about 20 min of training time. While the longest time is 100 epochs with a training time of about 2 h.

4.2 Testing Evaluation Stage

In the training process, the model that we have trained is stored for use in the testing phase. This stage aims to measure how accurate the number of epochs that we have trained is. The benchmark for whether or not the number of epochs used is determined by the resulting test accuracy value. Based on Table 3, the best number of epochs is produced by 75 epochs with a test accuracy of 85.7%. Followed by 50 epochs 75% and 100 epochs 82.3%. Then its performance will be measured with a confusion matrix. Comparison of the confusion matrix in each epoch is shown in Fig. 14.

Table 4. Model Performance

Number of Epochs	Accuracy	Precission	Recall
Epoch 50	75%	75%	75,25%
Epoch 75	85,67%	85,71%	85,71%
Epoch 100	82,3%	82,3%	82,3%

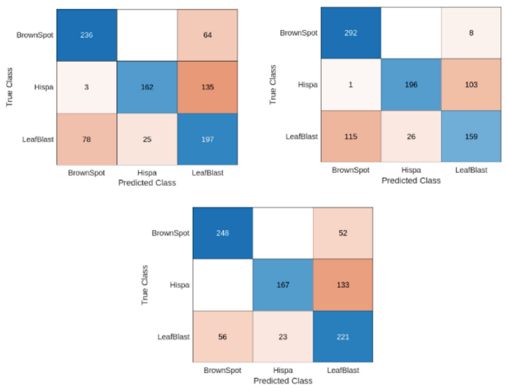

Fig. 14. The Confusion Matrix of Three Class

After we know the TP, TN, FP, and FN values of each class, the next step is to calculate Accuracy (1), Precision (2), and Recall (3). In this study, we use Accuracy, Precision, and Recall as benchmarks of epoch count performance. Ssuccessively 50 epochs, 75 epochs, 100 epochs.

Based on the Table 4, it can be seen that the best number of values is produced by 75 epochs, with an accuracy of 85.67%, a precision of 85.71% and a recall of 85.71%. Accuracy shows the accuracy of the number of epochs used, which means that the greater the accuracy, the smaller the error. Precision indicates how much data that belongs to one class is incorrectly classified as another class. The greater the precision value, the smaller the error that is 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

Rice plants are rice producers to consume food and nutritional needs for humans. However, plant-disturbing organisms (OPT) often attack rice plants, especially leaves, causing crop failure and disturbing human welfare. In this study, we apply deep learning to classify the types of diseases that attack rice leaves. As a comparison, we used 3 different number of epochs to see which one produces the highest accuracy value. Based on the research we were concluded that 75 epochs were the most accurate, reaching 85.67%.

So that we can detect diseases that attack the leaves of rice plants. The training phase aims to train the machine, then the testing phase produces performance from the training data with a specified number of epochs.

Acknowledgment. I would like thanks to PUI-PT Combinatorics and Graph, CGANT University of Jember who have assisted in this research. I also thank to LP2M University of Jember, Indonesia.

References

1. Alidrus, Syaiful Anam, Musthafa Aziz, Oddy Virgantara. 2021. Detection of Disease in Rice Plant Leaves using the Method *Convolutional Neural Networks.* Ponorogo : University of Darussalam Gontor.
2. Rice Cultivation (December 2019). http://cybex.pertanian.go.id/mobile/artikel/88796/BUDIDAYA-TANAMAN-PADI/
3. Y. Kawasaki, H. Uga, S. Kagiwada, and H. Iyatomi, "Basic Study of Automated Diagnosis of Viral Plant Diseases Using Convolutional Neural Networks" in Springer International Publishing Switzerland 2015., DOI: https://doi.org/10.1007/978-3-319-27863-6_59
4. Dafik, Z. R. Ridlo, I. H. Agustin, R. I. Baihaki, F. G. Febrianto, 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 Irigation System on Vertical Farming" in Press.
5. 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
6. R. C. Gonzales and R. E. Woods, Digital Image Processing Fourth Edition". England: Pearson Education, 2018.
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. M. Sardogan, A. Tuncer, and Y. Ozen, "Plant Leaf Disease Detection and Classification Based on CNN with LVQ Algorithm" in 3 rd International Conference on Computer Science and Engineering, 2018.
9. Ratnawati, L., & Sulistyaningrum, D. R. (2020). Application of Random Forest to Measure Disease Severity in Apple Leaves. ITS Journal of Science and Arts, 8(2), A71- A77.
10. 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.