Edelweiss Flower Species Classification Using Convolutional Neural Network with Adaptive Moment Estimation Optimizer (Adam)

Wibi Anto, Herlina Napitupulu, and Nurul Gusriani

Abstract—**Indonesia, as a tropical country, is rich in biodiversity. To ensure the sustainability of its natural resources, the government enforces regulations for the protection of plants and animals, particularly those classified as endemic species, which are organisms found only in specific regions. One such species is Anaphalis javanica, an endemic edelweiss species in Indonesia. Various conservation efforts have been implemented, including the establishment of laws, regulations, and designated conservation areas. Edelweiss flowers exhibit significant diversity in color and shape, with additional variations arising from differences in the treatment of wild and cultivated flowers. This diversity presents a challenge in accurately identifying edelweiss species, which requires high precision to avoid legal violations. To address this issue, this study employs a Convolutional Neural Network (CNN) with the Adaptive Moment Estimation (Adam) optimizer for the classification of edelweiss flower species. The research utilizes a dataset comprising 3498 training images, which are further divided into training and validation sets, and 1050 test images, categorized into three classes: Anaphalis javanica, Leontopodium alpinum, and Leucogenes grandiceps. The primary objective of this study is to develop a CNN architecture and program code optimized with Adam to classify edelweiss flower species and evaluate the model's performance across different learning rates. Additionally, the study compares the performance of the custom CNN model with pre-trained models, including MobileNetV2, ResNet50, and VGG19, to assess the effectiveness of transfer learning in improving classification accuracy.**

Index Terms—**Adam optimizer, CNN, Edelweiss flower, Image Classification**

I. Introduction

INDONESIA is a tropical country renowned for its rich biodiversity. To preserve its diverse natural resources, the government has established various conservation areas, including national parks, botanical forest parks, and nature tourism parks. According to UU No.5 of 1990, national parks are designated as nature conservation areas with original ecosystems, managed through a zoning system for purposes such as research, education, tourism, and recreation. Each national park has unique characteristics based on its geographical location, resulting in biodiversity that is often exclusive to specific regions. One such example is the Edelweiss flower.

The Directorate General of Natural Resources and Ecosystem Conservation has identified that Edelweiss flowers thrive exclusively in mountainous regions with full sunlight. Through Permen LHK No. P.106/Menlhk/Setjen/Kum.1/12/2018, the government declared Anaphalis javanica, an endemic Edelweiss species, as a protected plant species.

Accurate classification of Edelweiss flowers is essential to determine whether a flower belongs to a protected species, thereby avoiding legal violations and severe penalties. However, the diversity in Edelweiss species, influenced by variations in color, shape, and cultivation practices, poses a significant challenge for accurate identification. This challenge can be addressed through advancements in Machine Learning, particularly in the field of Computer Vision. Image Classification using Convolutional Neural Networks (CNNs) has proven effective in recognizing complex visual patterns and features, enabling precise differentiation of Edelweiss flower species.

To ensure the novelty of this study, a Systematic Literature Review (SLR) was conducted using the PRISMA method. The Publish or Perish tool was employed to search for metadata with the keywords: 'edelweiss' AND (CNN OR "convolutional neural networks"). The search yielded 169 results, which were filtered for duplicates and relevance using Python programming. After screening titles and abstracts, 17 articles were shortlisted, and further evaluation revealed only two articles with partial relevance. However, a detailed review of their full content indicated limited applicability to this study. Previous research on Edelweiss flower classification, such as the study on classification using Data Augmentation and Linear Discriminant Analysis [1], highlights the scarcity of studies in this domain. This underscores the need for further exploration into the classification of Edelweiss flower images.

Manuscript received September 27,2024; revised May 18, 2025. This work was supported in part by Universitas Padjadjaran through DRPM Unpad.

Wibi Anto is a bachelor's graduate from the Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Padjadjaran, West Java, Indonesia (e-mail:wibi20001@mail.unpad.ac.id).

Herlina Napitupulu is a lecturer in Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Padjadjaran, West Java, Indonesia (corresponding author, phone: +6281313315135; e-mail: herlina@unpad.ac.id).

Nurul Gusriani is a lecturer in Department of Mathematics, Faculty of Mathematics and Natural Sciences, Universitas Padjadjaran, West Java, Indonesia (e-mail: nurul.gusriani@unpad.ac.id)

II. Literature Review

A. Species Classification in Biology

Species classification in biological science considers several things including naming and identification, morphological and genetic characteristics, to using special techniques such as DNA analysis [2] . Classification techniques based on morphological characteristics are very likely to be carried out using only images with certain techniques.

B. Image Classification

Image classification is the process of categorizing and labeling a set of pixels or vectors of an image based on certain rules [3]. Some methods that are often used in image classification include Support Vector Machine (SVM), Convolutional Neural Network (CNN), faster RCNN [4], to transfer learning methods using pre-trained models [5].

C. Tensor

A tensor is an array of numbers arranged on a regular grid with an adjustable number of axes (dimensions) [6]. A tensor in programming is an array with dimensions which also means it is an extension of a matrix which is an array with two dimensions [7]. Suppose $\mathbf{X}$ is a tensor with three axes/dimensions then the element of $\mathbf{X}$ at coordinates m, n, p is $x_{m,n,p}$. Mathematically, the tensor is expressed as follows.

$$\mathbf{X} = \begin{bmatrix} [x_{1,1,1} x_{1,1,2} \cdots x_{1,1,s}] & [x_{1,2,1} x_{1,2,2} \cdots x_{1,2,s}] & \cdots & [x_{1,r,1} x_{1,r,2} \cdots x_{1,r,s}] \\ [x_{2,1,1} x_{2,1,2} \cdots x_{2,1,s}] & [x_{2,2,1} x_{2,2,2} \cdots x_{2,2,s}] & \cdots & [x_{2,r,1} x_{2,r,2} \cdots x_{2,r,s}] \\ \vdots & \vdots & \ddots & \vdots \\ [x_{q,1,1} x_{q,1,2} \cdots x_{q,1,s}] & [x_{q,2,1} x_{q,2,2} \cdots x_{q,2,s}] & \cdots & [x_{q,r,1} x_{q,r,2} \cdots x_{q,r,s}] \end{bmatrix}, \mathbf{X} \in \mathbb{R}^{m,n,p} \tag{1}$$

D. Image Processing

Image preprocessing is a method to transform raw image data into a clean image data, as most of the raw image data contain noise and contain some missing values or incomplete values, inconsistent values, and false [8]. Raw data can be overcome in several ways e.g. image correction, image enhancement, image restoration, and image compression [9]. The following are some of the image preprocessing techniques used in this research:

1) Image compression is used to reduce the size of an image either by reducing the dimensions or the quality of the image. Image compression can be done using the PIL library in Python.
2) Normalization to normalize pixel value from range of 0-255 to 0-1.
3) Mini batch division with TensorFlow.

Suppose X is a dataset with m data, $\mathbf{X_i}$ is the i^{th} data from dataset X in the form of a tensor and bs is the batch size, then the number of iteration processes can be calculated using the following equation.

$$T = \lceil \frac{m}{bs} \rceil \tag{2}$$

The mini batch is denoted by $X^{\{t\}}$ and can be expressed as follows

$$X^{\{t\}} = \{\mathbf{X}_{(bs\cdot(t-1)+1)}, \mathbf{X}_{(bs\cdot(t-1)+2)}, \ldots, \mathbf{X}_{(bs\cdot t)}\} \tag{3}$$

were $t = 1, 2, 3, \ldots T$

E. Convolutional Neural Network

Convolutional Neural Network (CNN) is an artificial neural network with a convolution process that emerged from the study of the visual cortex of the brain and has been used since the 1980s [10]. The convolution process is a mathematical calculation used to perform feature extraction on images [11]. CNN architecture generally consists of three types of layers: convolution layer, pooling layer, and fully connected layer. There are two types of propagation in CNN which are forward propagation and backpropagation. Forward propagation aims to forward the predicted output from the input layer by performing certain calculations from the input to the next layer while backpropagation aims to update the parameter values (weights and biases) by calculating the error of the predicted output [12].

F. Transfer Learning

Transfer learning is a concept where we train a model on one problem and then we can fine-tune and apply it on another similar kind of problem [13]. We compare the performance of the model using transfer learning with the CNN custom model that we built ourselves. In this research, we use transfer learning to improve the performance of the model by using a pre-trained model such as MobileNetV2 [14], ResNet50 [15], and VGG19 [16].

The transfer learning models are used as feature extractors by retaining the weights from the pre-trained models. The pre-trained models extract features from the input images, and the extracted features are then passed to custom dense layers for classification. This approach leverages the knowledge learned by the pre-trained models on large datasets, improving the performance of the classification task on the edelweiss flower species dataset.

G. Two-Dimensional Convolution Layer

A two-dimensional convolution layer is a layer of neurons with convolution operations. The TensorFlow library has a Conv2D function to create a two-dimensional convolution layer. There are hyperparameters in the Conv2D function such as filters, kernel size, activation, and strides [17].

The Conv2D function has a strides parameter that indicates the amount of pixel displacement after the convolution operation. A stride value of 1 means that every time a convolution operation is performed, there is a displacement of one pixel for the next convolution operation. The feature map resulting from the convolution operation at the L^{th} layer on a tensor $\mathbf{X}$ and filter $\mathbf{W}$ with bias β at layer L is given as follows.

$$\mathbf{z}^{(L)} = \text{Conv}(\mathbf{X}, \mathbf{W}^{(L)}) + \boldsymbol{\beta}^{(L)} \tag{4}$$

where the two-dimensional convolution operation with one filter and C color channels is shown as follows.

$$\text{Conv}(\mathbf{X}, \mathbf{W}) = y_{(q,r,s)} = \sum_{c=1}^{K_3} \sum_{a=1}^{K_1} \sum_{b=1}^{K_2} x_{(q+a-1,r+b-1,s+c-1)} \cdot w_{(a,b,c)} \tag{5}$$

where $y_{(q,r,s)}$ is the convolution result of the q^{th} row, r^{th} column, and s^{th} color channel, $\mathbf{X}$ is the input feature map, and $\mathbf{W}$ is the filter with dimensions (K_1, K_2, K_3).

The activation function often used in the convolution layer is the Rectified Linear Unit (ReLU), which processes the z_c value by converting every negative value to zero [18]. It can be mathematically expressed as $f(x) = \max(0, x)$

$$A_{(q,r,s)} = \alpha_{\text{ReLU}}(zc^{(L)}) = \max(0, zc^{(L)}) \tag{6}$$

So, the output of the L^{th} Conv2D layer can be seen as follows.

$$\mathbf{A}^{(L)} = \begin{bmatrix} [A_{1,1,1}A_{1,1,2}\dots A_{1,1,s}] & [A_{1,2,1}A_{1,2,2}\dots A_{1,2,s}] & \dots & [A_{1,r,1}A_{1,r,2}\dots A_{1,r,s}] \\ [A_{2,1,1}A_{2,1,2}\dots A_{2,1,s}] & [A_{2,2,1}A_{2,2,2}\dots A_{2,2,s}] & \dots & [A_{2,r,1}A_{2,r,2}\dots A_{2,r,s}] \\ \vdots & \vdots & \ddots & \vdots \\ [A_{q,1,1}A_{q,1,2}\dots A_{q,1,s}] & [A_{q,2,1}A_{q,2,2}\dots A_{q,2,s}] & \dots & [A_{q,r,1}A_{q,r,2}\dots A_{q,r,s}] \end{bmatrix} \tag{7}$$

H. Two-dimensional Pooling Layer

Pooling Layer is a layer of neurons with a merging operation to reduce the dimension by combining values in certain areas according to the filter size. There are two types of pooling operations which are max pooling and average pooling [18]. The max pooling operation is performed by taking the maximum value from a set of pixel values in a particular area. Let $\mathbf{X}$ be the input tensor in the pooling layer then the output of the L^{th} pooling layer can be expressed as follow.

$$\mathbf{P}^{(L)} = \text{MaxPooling2D}(\mathbf{X}) \tag{8}$$

I. Flatten Layer

Flatten layer is a layer of neurons that has the operation to flatten the dimension of the tensor input that enters the layer. Suppose there is a tensor $\mathbf{X}$ with dimensions (Q, R, S), then the output of the flatten layer can be expressed as follows.

$$FL(\mathbf{X}) = \vec{zf} = [x_{1,1,1}x_{1,2,1}x_{2,1,1}\dots x_{q,r,s}\dots x_{Q,R,S}] \tag{9}$$

J. Dense Layer

Dense layer is a layer of neurons that receive input from all neurons in the previous layer [19]. The last dense layer is used to classify between the classes in the dataset [20]. Mathematically, the output of each dense layer is expressed as follows.

$$zd_j^{(L)} = \sum_{k=1}^{N} w_{jk}^{(L)} \cdot x_k^{(L-1)} + \beta_j^{(L)} \tag{10}$$

Where:

- $zd_j^{(L)}$ is the value of the j-th neuron in the L-th layer.
- $w_{jk}^{(L)}$ is the weight between the j-th neuron in the L-th layer and the k-th neuron in the previous layer.
- $x_k^{(L-1)}$ is the input value for the dense layer of the neuron in the previous layer.
- N is the number of neurons in the previous layer.
- $\beta_j^{(L)}$ is the bias on the j-th neuron in the L-th layer.

In general, the activation function equation is expressed as follows:

$$d_j = \alpha(zd_j^{(L)}) \tag{11}$$

So that the output of the L^{th} dense layer with activation function α, having as many as N neurons, can be expressed as the following row vector:

$$\vec{d}^{(L)} = [d_1\, d_2\, \dots\, d_j\, \dots\, d_N] \tag{12}$$

There are two activation functions used, namely ReLU and Softmax. Respectively, the ReLU and Softmax functions are expressed in the following equations.

$$\alpha_{\text{ReLU}}\left(zd_j^{(L)}\right) = \max\left(0, zd_j^{(L)}\right) \tag{13}$$

$$\alpha_{\text{sm}}\left(zd_j^{(L)}\right) = \frac{e^{z_j^{(L)}}}{\sum_{j=1}^{n} e^{z_j^{(L)}}} \tag{14}$$

K. Optimizer

Optimizer is an algorithm used to minimize the loss function or to improve model performance by changing model parameters such as weights or learning rate [21]. Some optimizers include AdaGrad, RMSProp, SGDNesterov, AdaDelta, and Adam. Adam is robust and suitable for a wide variety of non-convex optimization problems in the machine learning field [22].

1) Loss function The loss function used for multiclass classification problems is Categorical crossentropy. Let $\hat{y}$ be the prediction result of the model and y is the value of the actual label in the train data, then the loss function of categorical cross-entropy is expressed as follows.

$$\text{Loss}(\hat{y}, y) = -\sum_{j=1}^{n} y_j \cdot \log(\hat{y}_j) \tag{15}$$

where $j = 1, 2, \dots, n$ with n is the number of classes of the dataset we used.

2) Cost function The cost function is used to find the average loss value by calculating each loss value obtained from each prediction result and each label y in the training data. Suppose there are as many as m prediction results and y labels, then the cost function can be expressed as follows:

$$J = \frac{1}{m}\sum_{p=1}^{m} \text{Loss}\left(\hat{y}_j^{(p)}, y_j^{(p)}\right) \tag{16}$$

where $p = 1, 2, \dots, m$.

3) Adam Algorithm Suppose there is a function J which is a cost function differentiable with respect to the model parameter θ and the initial values $m_t = v_t = 0$. Then the Adam update rule is as follows:

$$g_t(\theta) = \frac{\partial}{\partial\theta} J \tag{17}$$

$$m_t = \gamma_1 \cdot m_{t-1} + (1 - \gamma_1) \cdot g_t \tag{18}$$

$$v_t = \gamma_2 \cdot v_{t-1} + (1 - \gamma_2) \cdot g_t^2 \tag{19}$$

$$\hat{m}_t = \frac{m_t}{1 - \gamma_1^t} \tag{20}$$

$$\hat{v}_t = \frac{v_t}{1 - \gamma_2^t} \tag{21}$$

$$\theta_t = \theta_{t-1} - \text{lr}_i \cdot \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{22}$$

4) Accuracy The metric used to measure model performance is the accuracy metric with the following formula.

$$Acc = \frac{true\ prediction}{total\ prediction} \tag{23}$$

L. Python

Python is a programming language created by Guido van Rossum that was first released in 1991. It is a high-level programming language that supports a variety of programming paradigms, including object-oriented programming, functional programming, and procedural programming and is popular because it is easy to understand [23]. Some of the modules and libraries used in this research include the following.

1) TensorFlow
2) Matplotlib
3) Python Imaging Library(PIL)
4) modul CSV

III. Object and Research Method

A. Object

The object of this research is an image dataset of edelweiss flower species, consisting of 3498 training images and 1050 test images. The dataset is categorized into three species: Anaphalis javanica, Leontopodium alpinum, and Leucogenes grandiceps [24].

B. Research Method

Custom CNN Model

This study employs a Convolutional Neural Network (CNN) with the Adaptive Moment Estimation (Adam) optimizer to classify edelweiss flower species using image data. The dataset is divided into three classes based on the species. The research methodology is structured into several stages, as outlined below:

1) **Image Preprocessing** Image preprocessing involves compressing the images, resizing them, converting them into tensors, and normalizing pixel values. The compression process reduces the image size to a maximum width of 512 pixels while maintaining the aspect ratio and reducing the image quality to 85% using the PIL library. The images are then resized to 256×256, normalized by dividing each pixel value by 255, and converted into tensors. The training dataset is split into training and validation sets with an 8:2 ratio. Finally, the dataset is divided into mini-batches with a batch size of 32.
2) **Learning Rate Initialization** The learning rate is initialized as $lr = 10^{-i}$, where $i = 1$.
3) **Model Training** The architecture of the custom CNN model consists of input layer, hidden layer: 12 convolutional layers and 12 pooling layers, one flatten layer, and two dense layers. The first convolutional layer uses a filter size of 3×3 with a stride of 1, while subsequent convolutional layers use a filter size of 2×2 with a stride of 2. The pooling layers employ max pooling with a pool size of 2×2 and a stride of 2. The ReLU activation function is used in the convolutional layers, while the final dense layer uses the Softmax activation function. The model is compiled using the Adam optimizer with the initialized learning rate. The model architecture is illustrated in Fig. 1.
   a) **Forward Propagation** Forward propagation begins by inputting the first mini-batch tensor into the first convolutional layer. The process involves applying convolutional operations, pooling, and activation functions sequentially across all layers until the final dense layer produces a classification result.
   b) **Backpropagation** Backpropagation is performed after obtaining the classification result. The loss value is calculated using the categorical cross-entropy loss function, and the model parameters (weights and biases) are updated using the Adam optimizer. This process is repeated for all mini-batches in the training dataset across 20 epochs.

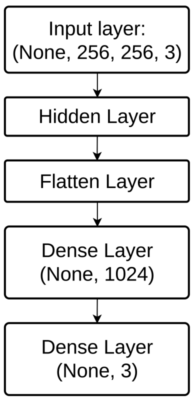


Fig. 1: CNN custom model architecture

4) **Model Testing** After training for 20 epochs, the model is tested using the test dataset, which is divided into mini-batches. The model's performance, including accuracy and cost, is recorded for each learning rate.
5) **Learning Rate Update** The learning rate is updated to $lr = 10^{-i}$, where $i := i + 1$, and the training process is repeated until $i > 4$.

Transfer Learning Model

The custom CNN model is compared with transfer learning models using pre-trained architectures such as MobileNetV2, ResNet50, and VGG19. The methodology for building and training the transfer learning models is as follows:

1) **Image Preprocessing** The image preprocessing steps are identical to those used for the custom CNN model.
2) **Learning Rate Initialization** The learning rate is initialized as $lr = 10^{-4}$.
3) **Model Training** The transfer learning model architecture is constructed by integrating the pre-trained model as a feature extractor, followed by a flatten layer and dense layers for classification. The architecture is shown in Fig. 2.

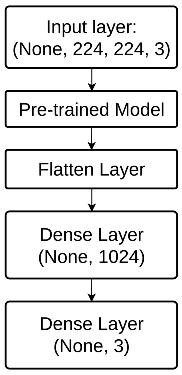


Fig. 2: Transfer learning model architecture

   a) **Forward Propagation** The input tensor is passed through the pre-trained model to extract features, which are then processed by the flatten and dense layers to produce classification results.

b) **Backpropagation** The backpropagation process is similar to that of the custom CNN model, with the Adam optimizer used to update the model parameters.

4) **Model Testing** After the model has been trained for 10 epochs or has achieved 100% accuracy on both the training and testing datasets, the testing process is conducted using the test dataset, which is divided into mini-batches with a batch size of 32.

IV. Result and Discussion

A. Matematical Computation for Custom CNN Model

Image preprocessing

Image preprocessing is done by compressing the image, resizing, transforming to tensor, and normalizing the pixel value. Compress the image using the help of the PIL library by reducing the size and quality of the image. The comparison of images before and after the compression process can be seen in Fig. 3.

Fig. 3: Image compression comparison

Comparison of the image before compression (left) and after compression (right). The original image size is 1600px × 1200px with a memory size of 4.97MB, while the compressed image is resized to 512px × 384px with a memory size of 396KB.

Resize, transform image to tensor, and normalize the pixel value using the ImageDataGenerator module. The comparison of images before and after resizing are shown in Fig. 4.

Fig. 4: Image resizing comparison

Comparison of the images before resizing (left) and after resizing (right). Transform the image into tensor and normalize the pixel values. Tensor before normalization:

$$\mathbf{X} = \begin{bmatrix} [128\ 145\ 127] & [127\ 128\ 95] & \dots & [137\ 107\ 76] \\ \vdots & \vdots & \ddots & \vdots \\ [94\ 101\ 64] & [106\ 101\ 91] & \dots & [116\ 98\ 74] \\ \vdots & \vdots & \ddots & \vdots \\ [155\ 167\ 134] & [149\ 169\ 142] & \dots & [44\ 46\ 45] \end{bmatrix} \tag{24}$$

Tensor after normalization:

$$\mathbf{X} = \begin{bmatrix} [0.502\ 0.569\ 0.498] & [0.498\ 0.502\ 0.373] & \dots & [0.537\ 0.42\ 0298] \\ \vdots & \vdots & \ddots & \vdots \\ [0.369\ 0.396\ 0.251] & [0.416\ 0.396\ 0.357] & \dots & [0.455\ 0.384\ 029] \\ \vdots & \vdots & \ddots & \vdots \\ [0.608\ 0.655\ 0.525] & [0.584\ 0.663\ 0.557] & \dots & [0.173\ 0.18\ 0176] \end{bmatrix} \tag{25}$$

Split the train dataset into train and validation datasets then divide into mini batches. The train data consists of 2799 data so that using equations (2) is obtained:

$T = \lceil \frac{2799}{32} \rceil = 88$

so that each mini batch in the train dataset becomes:

$$X_{tn}^{\{t\}} = \{\mathbf{X}_{(32\cdot(t-1)+1)}, \mathbf{X}_{(32\cdot(t-1)+2)}, \dots, \mathbf{X}_{(32\cdot t)}\}$$

$$t = 1, 2, \dots, 88$$

Let $t = 1$, thus obtained:

$$X_{tn}^{1} = \{X_1, X_2, \dots, X_{32}\}$$

Training model

The model training process with the first learning rate value is $10^{(-i)}$ where $i = 1$ which is done for 20 epochs. Each epoch processes the entire mini batch through the forward propagation process and the backpropagation process to obtain the cost value and update the model parameters. Select $\mathbf{Xtn}_1^{\{1\}}$ substitute to the first convolution with filter $\mathbf{W}$ using equation (4) so that it is obtained:

$$zc^{(1)} = Conv(Xtn_1^{\{1\}}, W^{(1)}) + \beta^{(1)}$$

select the first filter of $W^{(1)}$ thus obtained:

$$zc^{(1)} = \begin{bmatrix} [0.502\ 0.569\ 0.498] & [0.498\ 0.502\ 0.373] & \dots & [0.537\ 0.42\ 0.298] \\ \vdots & \vdots & \ddots & \vdots \\ [0.369\ 0.396\ 0.251] & [0.416\ 0.396\ 0.357] & \dots & [0.455\ 0.384\ 0.29] \\ \vdots & \vdots & \ddots & \vdots \\ [0.608\ 0.655\ 0.525] & [0.584\ 0.663\ 0.557] & \dots & [0.173\ 0.18\ 0.176] \end{bmatrix} *$$

$$\begin{bmatrix} [-0.079\ -0.075\ -0.187] & [0.07\ 0.039\ -0.137] & \dots & [-0.169\ -0.175\ -0.155] \\ \vdots & \vdots & \ddots & \vdots \\ [0.112\ 0.025\ 0.007] & [-0.164\ -0.131\ -0.11] & \dots & [0.151\ 0.142\ -0.168] \\ \vdots & \vdots & \ddots & \vdots \\ [0.04\ 0.038\ 0.025] & [-0.15\ 0.135\ -0.039] & \dots & [-0.095\ 0.067\ 0.130] \end{bmatrix} + 0$$

select the input feature map position 1 of $\mathbf{Xtn}_1^{\{1\}}$, perform the convolution operation using equation (5) to obtain:

$$A_{1,1,1}^{(1)} = (x_1^{(1)})_{1,1,1} \cdot w_{1,1,1} + (x_1^{(1)})_{1,2,1} \cdot w_{1,2,1} + (x_1^{(1)})_{1,3,1} \cdot w_{1,3,1}$$

$$+ \cdots + (x_1^{(1)})_{3,2,3} \cdot w_{3,2,3} + (x_1^{(1)})_{3,3,3} \cdot w_{3,3,3}$$

$$
\begin{aligned}
A^{(1)}_{1,1,1} &= 0.502 \cdot (-0.079) + 0.498 \cdot 0.07 + 0.482 \cdot (-0.169) \\
&\quad + 0.557 \cdot (-0.039) + 0.71 \cdot 0.130 \\
A^{(1)}_{1,1,1} &= -0.018870000000000012
\end{aligned}
$$

substitute the value of $zc^{(1)}$ to the ReLU activation function using equation (6) to obtain:

$$A^{(1)}_{1,1,1} = max(0; -0,018870000000000012) = 0$$

Move the feature map $\mathbf{Xtn}^{\{1\}}_{1}$ by one pixel then repeat the convolution operation until the feature map is located at the last position and repeat the convolution process using the next filter until it is obtained:

$$
\mathbf{A}^{(1)} = \begin{bmatrix}
[0\ 0.204\ \ldots\ 0] & [0\ 0.231\ \ldots\ 0] & \ldots & [0\ 0.181\ \ldots\ 0] \\
\vdots & \vdots & \ddots & \vdots \\
[0\ 0.17\ \ldots\ 0] & [0\ 0.177\ \ldots\ 0] & \ldots & [0\ 0.124\ \ldots\ 0] \\
\vdots & \vdots & \ddots & \vdots \\
[0\ 0.059\ \ldots\ 0.005] & [0\ 0.093\ \ldots\ 0] & \ldots & [0\ 0.02\ \ldots\ 0]
\end{bmatrix} + 0
$$

Substitute $\mathbf{A}^{(1)}$ as input for the next layer, namely the pooling layer using equation (8) so that it is obtained:

$$\mathbf{P}^{(2)} MaxPooling2D(\mathbf{A}^{(1)})$$

$$
\mathbf{P}^{(2)} = \begin{bmatrix}
[0\ 0.231\ \ldots\ 0] & [0\ 0.194\ \ldots\ 0] & \ldots & [0\ 0.216\ \ldots\ 0] \\
\vdots & \vdots & \ddots & \vdots \\
[0\ 0.198\ \ldots\ 0.022] & [0\ 0.172\ \ldots\ 0] & \ldots & [0\ 0.172\ \ldots\ 0] \\
\vdots & \vdots & \ddots & \vdots \\
[0\ 0.072\ \ldots\ 0.016] & [0\ 0.059\ \ldots\ 0.01] & \ldots & [0\ 0.093\ \ldots\ 0]
\end{bmatrix}
$$

Output pooling layer $\mathbf{P}^{(2)}$ is used as input for the next layer, namely the second convolution layer and so on with the same calculation process until the 12th layer is obtained $\mathbf{P}^{(12)}$.

$$
\mathbf{P}^{(12)} = \begin{bmatrix}
[0.0412\ 0.0241 \ldots 0] & [0.0592\ 0.0226 \ldots 0] \\
[0.0368\ 0.0416 \ldots 0] & [0.0395\ 0.0406 \ldots 0]
\end{bmatrix}
$$

substitute $\mathbf{P}^{(12)}$ to equation (9) to obtain:

$$FL(\mathbf{P}^{(12)}) = \vec{zf} = [P^{(12)}_{1,1,1}\ P^{(12)}_{1,2,1}\ P^{(12)}_{2,1,1}\ P^{(12)}_{2,2,1}\ \ldots P^{(12)}_{2,2,256}\ ]$$

$$\vec{zf} = [0.0412\ 0.0241 \ldots 0 \ldots 0.0395\ 0.0406 \ldots 0]$$

substitute $\vec{zf}_k$ with $k = 1, 2, \ldots, 1024$ to equation (10) thus obtain:

$$zd^{(14)}_{j} = \sum_{k=1}^{1024} w^{(14)}_{jk} \cdot \vec{zf}_k + \beta^{(14)}_{j}$$

$$zd^{(14)}_{1} = \sum_{k=1}^{1024} w^{(14)}_{1k} \cdot \vec{zf}_k + \beta^{(14)}_{j}$$

$$
\begin{aligned}
z^{(14)}_{d_1} &= \Big(w^{(14)}_{11} \cdot \vec{z_f}_1 + w^{(14)}_{12} \cdot \vec{z_f}_2 + w^{(14)}_{13} \cdot \vec{z_f}_3 \\
&\quad + \cdots + w^{(14)}_{1\ 1024} \cdot \vec{z_f}_{1024}\Big) + 0 \\
z^{(14)}_{d_1} &= (0.0553 \cdot 0.0412 + 0.0366 \cdot 0.02407 + 0.0075 \cdot 0 \\
&\quad + \cdots + 0.0186 \cdot 0) + 0 \\
z^{(14)}_{d_1} &= 0.044526230544
\end{aligned}
$$

substitute the value of $zd^{(14)}_{1}$ to equation (13) thus obtain:

$$
\begin{aligned}
d_1 &= \max(0, z^{(14)}_{d_1}) \\
&= \max(0, 0.044526230544) \\
&= 0.044526230544
\end{aligned}
$$

do the calculation until $zd^{(14)}_{128}$ thus obtained d_j, $j = 2, 3, \ldots 128$. By equation 12, it is obtained:

$$\vec{d}^{(14)} = [d_1\ d_2\ d_3\ d_4\ \ldots d_128]$$

$$\vec{d}^{(14)} = [0.0445\ 0\ 0\ 0.01745\ \ldots 0.0633]$$

Substitute $\vec{d}^{(14)}$ to equation (11) thus obtain:

$$zd^{(15)}_{j} = \sum_{k=1}^{128} \left(w^{(15)}_{jk} \cdot d^{(14)}_{k}\right) + \beta^{(15)}_{j}$$

$$zd^{(15)}_{1} = \sum_{k=1}^{128} \left(w^{(15)}_{1k} \cdot d^{(14)}_{k}\right) + \beta^{(15)}_{1}$$

$$zd^{(15)}_{2} = \sum_{k=1}^{128} \left(w^{(15)}_{2k} \cdot d^{(14)}_{k}\right) + \beta^{(15)}_{2}$$

$$zd^{(15)}_{3} = \sum_{k=1}^{128} \left(w^{(15)}_{3k} \cdot d^{(14)}_{k}\right) + \beta^{(15)}_{3}$$

With an analogous calculation process to the previous dense layer calculation, it is then obtained:

$$\vec{z_d}^{(15)} = \begin{bmatrix}0.0835465 & 0.0114078 & -0.0050581\end{bmatrix}$$

Substitute $\vec{z_d}^{(15)}$ to the Softmax activation function in equation (14) to obtain: The calculation of d_j using the Softmax activation function is as follows:

$$d_j = \alpha_{\text{sm}}(z^{(15)}_{d_j}) = \frac{e^{z^{(15)}_{j}}}{\sum_{j=1}^{3} e^{z^{(15)}_{j}}}$$

$$
\begin{aligned}
d_1 &= \alpha_{\text{sm}}(z^{(15)}_{d_1}) = \frac{e^{z^{(15)}_{1}}}{e^{z^{(15)}_{1}} + e^{z^{(15)}_{2}} + e^{z^{(15)}_{3}}} \\
&= \frac{e^{0.083546534181}}{e^{0.083546534181} + e^{0.011407849379} + e^{-0.005058117211}} \\
&= 0.35141861
\end{aligned}
$$

Find the value of d_2 and d_3 with the same calculation process to obtain:

$$d_2 = 0.32696053, \quad d_3 = 0.32162088$$

Substitute the values of d_1, d_2, and d_3 to equation (13) to obtain:

$$
\begin{aligned}
\vec{d}^{(14)} &= [d_1\ d_2\ d_3] \\
&= [0.35141861\ 0.32696053\ 0.32162088]
\end{aligned}
$$

Based on the dataset, it is known that the image has the label Anaphalis javanica, so the actual label is as follows:

$$\vec{y} = \begin{bmatrix}1 & 0 & 0\end{bmatrix}$$

Substitute $\vec{y}$ and $\vec{d}^{(15)}$ to equation (15) to obtain:

$$
\begin{aligned}
&\text{Loss}(\vec{d}^{(15)}, \vec{y}) = -\textstyle\sum_{j=1}^{3} y_j \cdot \log(d_j) \\
&= -(1 \cdot \log(0.351418614388) + 0 \cdot \log(0.32696053) \\
&+ 0 \cdot \log(0.32162088))
\end{aligned}
$$

$= -\log(0.351418614388)$
$= 1.0457771325547065$

Select the next tensor from the mini batch $Xtn^{\{1\}}$, which is $\mathbf{Xtn}_p^{\{1\}}$ where $p = 2, 3, \ldots, 32$. Then do the forward propagation process from the beginning until getting the loss function value of each tensor classification result in the mini batch $\mathbf{Xtn}_p^{\{1\}}$. Substitute $Loss^p(\vec{d}^{(15)}, \hat{y})$ to equation (16) is obtained:

$$\begin{aligned} J &= \frac{1}{32}\sum_{p=1}^{32} \text{Loss}(d_j^{(p)}, y_j^{(p)}) \\ &= \frac{1}{28}(1.04577 + 1.14437 + \cdots + 1.1138) \\ &= 1.0998 \end{aligned}$$

Based on forward propagation, the weight and bias of the CNN model are obtained as follows:

$$\mu = [\mathbf{W}^{(1)}, \beta^{(1)}, \mathbf{W}^{(3)}, \beta^{(3)}, \mathbf{W}^{(5)}, \beta^{(5)}, \mathbf{W}^{(7)}, \beta^{(7)}, \mathbf{W}^{(9)}, \beta^{(9)}, \mathbf{W}^{(11)}, \beta^{(11)}, w_{jk}^{(14)}, \beta^{(14)}, w_{jk}^{(15)}, \beta^{(15)}]$$

Select $\mu_{16} = \beta^{(15)}$, where:

$$\beta^{(15)} = [0, 0, 0]$$

Select $\beta_1^{(15)}$, substitute $\beta_1^{(15)}$ to equation (17) to obtain the gradient value of the bias parameter at the 15th layer in the first mini-batch iteration as follows:

$$g_1(\beta_1^{(15)}) = \frac{\partial}{\partial \beta_1^{(15)}} J$$

by using the tf.GradientTape() function, the obtained value of g_1 is as follows:

$$g_1(\beta_1^{(15)}) = 1,28363121$$

substitute g_1 to equation (18) and equation (19) to obtain:

$$\begin{aligned} m_1 &= \gamma_1 \cdot m_0 + (1 - \gamma_1) \cdot g_1 \\ &= 0.9 \cdot 0 + (1 - 0.9) \cdot (1.28363121) \\ &= 0.128363118 \\ v_1 &= \gamma_2 \cdot v_0 + (1 - \gamma_2) \cdot g_1^2 \\ &= 0.999 \cdot 0 + (1 - 0.999) \cdot (1.28363121)^2 \\ &= 0.00164770917 \end{aligned}$$

Substitute m_1 to equation (20) to obtain:

$$\begin{aligned} \hat{m}_1 &= \frac{0.128363118}{1 - 0.9} \\ &= 1.28363121 \end{aligned}$$

Substitute v_1 to equation (21) to obtain:

$$\begin{aligned} \hat{v}_1 &= \frac{0.00164770917}{1 - 0.999} \\ &= 1.64770913 \end{aligned}$$

Substitution $\hat{m}_1$ and $\hat{v}_1$ to equation (22) to obtain:

$$\begin{aligned} \left(\beta_1^{(15)}\right)_1 &= \left(\beta_1^{(15)}\right)_0 - 0.1 \cdot \frac{1.28363121}{\sqrt{1.64770913 + 10^{-8}}} \\ &= -0.001 \end{aligned}$$

Perform the parameter update process using equation (17) to equation (22) repeatedly until all model parameters have been updated. Continue the model training process to the next iteration $t := t + 1$ by using the next mini batch to train the model. Test the model using the validation dataset. Testing is done using the validation dataset to get the cost and accuracy values. Training the model is done up to 20 epochs then save the model and model performance results.

Testing model

Model testing is done using the test dataset. Change the dataset into mini batches as in the train dataset then find the cost and accuracy by using each mini batch as input data for the model.

B. *Model Performance*

The model training process is performed iteratively using the same model architecture but with different learning rate values. Learning rate with a value of $10^{(-i)}$ where $i \leq 4$ will be updated every time the model has been saved and tested using the test dataset. Model performance is calculated during training and testing.

1) Training Performance

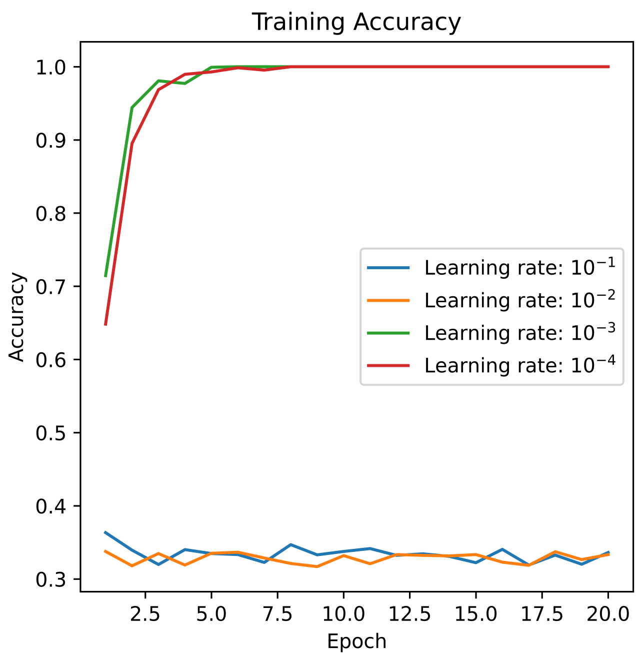
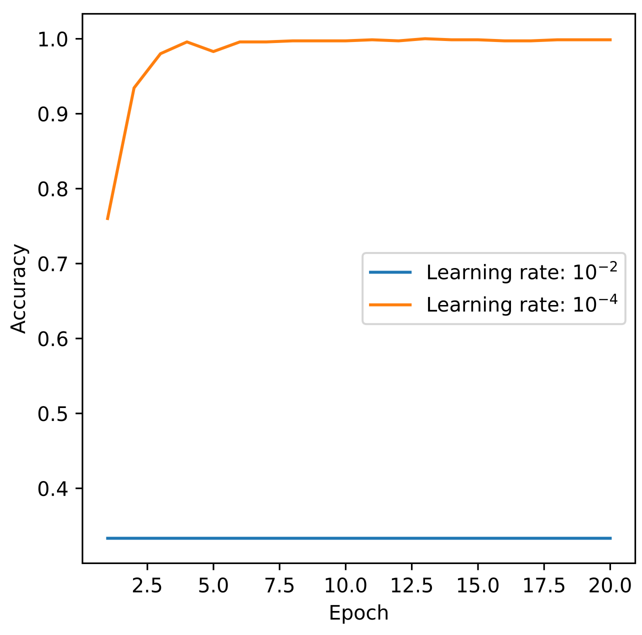


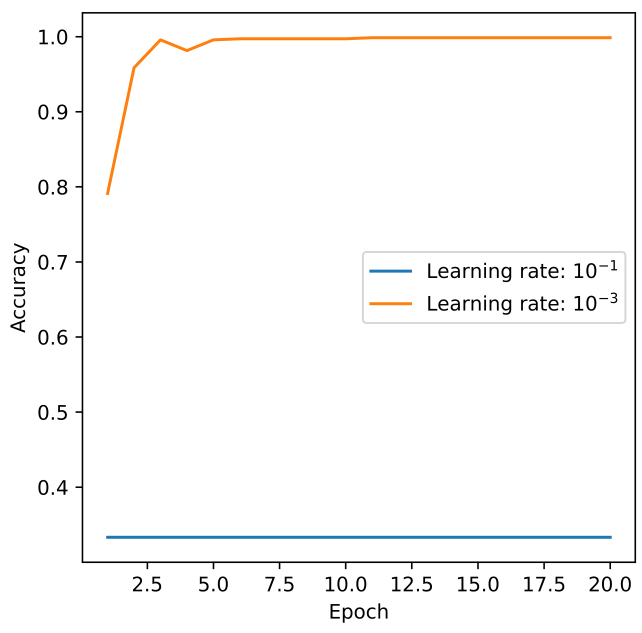
Fig. 5: Training accuracy graph for the training dataset

Based on Fig. 5, models with learning rates of 10^{-1} and 10^{-2} showed no improvement in accuracy, whereas models with learning rates of 10^{-3} and 10^{-4} demonstrated significant accuracy improvements. Similar to the training phase, models with learning rates of 10^{-1} and 10^{-2} showed no improvement in validation accuracy, with both models remaining at a constant value of 0.3333, whereas models with learning rates of 10^{-3} and 10^{-4} exhibited significant accuracy improvements (See Fig. 6).

2) Testing Performance After the model is trained up to 20 epochs and stored, the last step is to test the model using data that is not used during the training process, namely the test dataset. After testing the model with



(a) Validation accuracy for $lr = 10^{-2}, 10^{-4}$



(b) Validation accuracy for $lr = 10^{-1}, 10^{-3}$

Fig. 6: Validation accuracy comparisons for different learning rates

a learning rate of $10^{(-i)}$ where $i = 1, 2, 3, 4$ The classification results can be seen in TABLE I.

TABLE I: Classification results using the test dataset

Species	Actual Data	Classification Result (lr)			
		10^{-1}	10^{-2}	10^{-3}	10^{-4}
Anaphalis javanica	350	0	0	392	290
Leontopodium alpinum	350	0	1050	245	322
Leucogenes grandiceps	350	1050	0	413	438
Total	**1050**	**1050**	**1050**	**1050**	**1050**

Based on TABLE I, the model with learning rate 10^{-1} and 10^{-2} cannot perform classification well and the entire test dataset is classified in only one class while the model with 10^{-3} and can classify the test dataset into three different classes. The accuracy performance of the classification results on the test dataset can be seen in TABLE II.

TABLE II: Model performance on the test dataset

Species	Classification Result (lr)			
	10^{-1}	10^{-2}	10^{-3}	10^{-4}
True prediction	350	350	811	851
False prediction	700	700	239	199
Total prediction	1050	1050	1050	1050
Accuracy	**33,3%**	**33,3%**	**77,238%**	**81,048%**

Based on TABLE II, it is obtained that the best accuracy value is obtained by the model with $lr = 10^{-4}$. Model with $lr = 10^{-4}$ successfully classified 851 image data from 1050 test dataset images or 81.043%.

C. *Comparison with Transfer Learning Model*

The model performance of custom CNN Model is compared with transfer learning using pre-trained models such as MobileNetV2, ResNet50, and VGG19. The results of the classification accuracy of the three models can be seen in TABLE III.

TABLE III: Comparison of model performance

Model	Epoch (Training Acc 100%)	Test Accuracy	Trainable Params
Custom CNN Model	7	81.048%	15.2M
MobileNetV2	5	82.952%	64.3M
ResNet50	> 10	75.714%	102.7M
VGG19	2	98.000%	25.7M

Based on TABLE III, VGG19 achieved the highest test accuracy of 98.000% and was the fastest to reach 100% accuracy on both the training and validation datasets, requiring only 2 epochs. However, it also had the longest training time per epoch, taking up to 2830 seconds. In contrast, the Custom CNN model had the smallest number of trainable parameters (15.2M), making it the most lightweight model in terms of complexity, while still achieving a respectable test accuracy of 81.048%. The MobileNetV2 model achieved a test accuracy of 82.952% with a relatively small number of trainable parameters (64.3M) and reached 100% accuracy on both the training and validation datasets in just 5 epochs, compared to the custom CNN model, which required 7 epochs. The ResNet50 model, while having the highest number of trainable parameters (102.7M), achieved a test accuracy of 75.714% but required more than 10 epochs to reach 100% accuracy on the training dataset.

V. Conclusion

Based on the research conducted, the following conclusions can be drawn:

1) The custom CNN model was successfully developed using the Adaptive Moment Estimation (Adam) optimizer. It featured a lightweight architecture with only 15.2M trainable parameters, making it computationally efficient while achieving a respectable test accuracy of 81.048%.

2) Among the tested learning rates, 10^{-4} provided the best performance for the custom CNN model, achieving the highest accuracy of 81.048% on the test dataset.
3) Compared to pre-trained models, the custom CNN model demonstrated lower accuracy but excelled in simplicity and efficiency. While VGG19 achieved the highest test accuracy (98.000%) and MobileNetV2 reached 100% accuracy faster (5 epochs), the custom CNN model remains a viable option for resource-constrained environments due to its minimal computational requirements.

REFERENCES

[1] F. R. Malau and D. I. Mulyana, "Classification of edelweiss flowers using data augmentation and linear discriminant analysis methods," *Journal of Applied Engineering and Technological Science (JAETS)*, vol. 4, no. 1, pp. 139–148, 2022.

[2] V. T. Ho, T. K. P. Tran, T. T. T. Vu, and S. Widiarsih, "Comparison of matk and rbcl dna barcodes for genetic classification of jewel orchid accessions in vietnam," *Journal of Genetic Engineering and Biotechnology*, vol. 19, no. 1, p. 93, 2021.

[3] M. Shinozuka and B. Mansouri, "Synthetic aperture radar and remote sensing technologies for structural health monitoring of civil infrastructure systems," in *Structural health monitoring of civil infrastructure systems*. Elsevier, 2009, pp. 113–151.

[4] Y. Zhang, C. Song, and D. Zhang, "Deep learning-based object detection improvement for tomato disease," *IEEE access*, vol. 8, pp. 56 607–56 614, 2020.

[5] H. Bichri, A. Chergui, and M. Hain, "Image classification with transfer learning using a custom dataset: comparative study," *Procedia Computer Science*, vol. 220, pp. 48–54, 2023.

[6] I. Goodfellow, "Deep learning," 2016.

[7] H. Jean, "Deep learning book series - scalars, vectors, matrices, and tensors," 2021. [Online]. Available: https://hadrienj.github.io/posts/Deep-Learning-Book-Series-2.1-Scalars-Vectors-Matrices-and-Tensors/

[8] S. Chatterjee, S. Ghosh, S. Dawn, S. Hore, and N. Dey, "Forest type classification: a hybrid nn-ga model based approach," in *Information Systems Design and Intelligent Applications: Proceedings of Third International Conference INDIA 2016, Volume 3*. Springer, 2016, pp. 227–236.

[9] J. Chaki and N. Dey, *A beginner's guide to image preprocessing techniques*. CRC Press, 2018.

[10] A. Géron, *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*, 2nd ed. O'Reilly Media, Inc., 2019. [Online]. Available: https://www.oreilly.com/library/view/hands-on-machine-learning/9781492032632/

[11] N. Ketkar, J. Moolayil, N. Ketkar, and J. Moolayil, "Convolutional neural networks," *Deep learning with Python: learn best practices of deep learning models with PyTorch*, pp. 197–242, 2021.

[12] K. Gupta, "Forward propagation and back propagation simplified," 2024, accessed: 2024-05-31. [Online]. Available: https://medium.com/@kavita_gupta/forward-propagation-and-back-propagation-simplified-0b49f4e8732f

[13] J. Gupta, S. Pathak, and G. Kumar, "Deep learning (cnn) and transfer learning: a review," in *Journal of Physics: Conference Series*, vol. 2273, no. 1. IOP Publishing, 2022, p. 012029.

[14] M. Sandler, A. Howard, M. Zhu, A. Zhmoginov, and L.-C. Chen, " MobileNetV2: Inverted Residuals and Linear Bottlenecks ," in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. Los Alamitos, CA, USA: IEEE Computer Society, Jun. 2018, pp. 4510–4520. [Online]. Available: https://doi.ieeecomputersociety.org/10.1109/CVPR.2018.00474

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[17] T. Team, "tf.keras.layers.conv2d," 2024, accessed: 2024-05-03. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D

[18] B. Pang, E. Nijkamp, and Y. N. Wu, "Deep learning with tensorflow: A review," *Journal of Educational and Behavioral Statistics*, vol. 45, no. 2, pp. 227–248, 2020.

[19] P. Raja and P. Suresh, "Variety of ovarian cysts detection and classification using 2d convolutional neural network," *Multimedia Tools and Applications*, vol. 83, no. 16, pp. 49 473–49 491, 2024.

[20] G. Tripathi, K. Singh, and D. K. Vishwakarma, "Crowd emotion analysis using 2d convnets," in *2020 third international conference on smart systems and inventive technology (ICSSIT)*. IEEE, 2020, pp. 969–974.

[21] F. A. A. Said Aldemir and A. B. Kanburoğlu, "Effects of loss functions and optimizers on click-through rate prediction," in *10th International Conference on Advanced Technologies (ICAT'22)*, 2022, pp. 6–10.

[22] D. P. Kingma, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[23] A. Rawat, "A review on python programming," *International Journal of Research in Engineering, Science and Management*, vol. 3, no. 12, pp. 8–11, 2020.

[24] Ndomalau, "Edelweiss flower dataset," 2025. [Online]. Available: https://www.kaggle.com/datasets/ndomalau/edelweis-flower