International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

Research on Flower Classification Based on the Improved EfficientNetB7

Xiao Zuowen
School of Electronic Information and Electrical Engineering
Yangtze University
Jingzhou, China

Abstract: EfficientNet-B7 is an efficient deep convolutional neural network architecture, belonging to the EfficientNet series of models. This model, through systematic research on model scaling methods, proposes a compound scaling technique, and simultaneously optimizes the network's depth, width, and input resolution, thereby achieving a better balance between accuracy and computational efficiency. This paper elaborates on the working principle of EfficientNet-B7 in detail. A flower dataset with 104 categories was downloaded from Google Cloud Server, and EfficientNet-B7 was introduced to implement flower classification. To reduce model overfitting, the DropPath regularization term was added after the loss function. Through validation and testing, EfficientNet-B7 can effectively classify all flowers successfully, with a success rate reaching 100%. The addition of the DropPath regularization term can effectively reduce training time and improve network communication efficiency. Experiments show that the flower classification research based on EfficientNet-B7 is practical and effective, and it is of great significance to the study of flower classification and recognition.

Keywords: EfficientNet-B7; deep convolutional neural network; flower classification; regularization term

1. Introduction

Deep Convolutional Neural Networks (DCNN) are a type of deep learning architecture specifically designed for processing grid-structured data, such as images. By stacking multiple convolutional and pooling layers, DCNNs can automatically extract hierarchical features from data. In recent years, they have been widely applied in many significant fields, including image recognition and classification, video analysis, natural language processing, speech recognition, etc., and have driven the development of deep learning technology. Common deep convolutional neural networks include LeNet (one of the earliest convolutional neural networks, mainly used for handwritten character recognition, including convolutional layers, pooling layers, and fully connected layers, mainly used for simple image classification tasks), AlexNet (consisting of an 8-layer network, using ReLU activation functions and Dropout technology, significantly improving the accuracy of image classification, widely used in image recognition and classification tasks), and the EfficientNet-B7 mentioned in this paper, which is widely applied.

Studying the classification and recognition of flowers is of great significance to the study of plant classification and recognition. Currently, plant species classification and recognition are mainly achieved through botanical classification indexes and manual reference materials, which are both time-consuming and labor-intensive. With the continuous in-depth research and wide application of deep convolutional neural networks, new directions and ideas have been provided for the classification and recognition of plant species. In this paper, an improved EfficientNet-B7 neural network model for flower classification is proposed, which is helpful for the study of plant species classification and recognition.

2. EfficientNet-B7 Neural Network Model

Before EfficientNet, the development of Convolutional Neural Networks (CNNs) usually depended on a fixed resource budget, and then performance was improved by increasing the network's width (number of channels), depth (number of layers), or the resolution of the input images. However, these one-dimensional scaling methods often lead to a significant increase in the number of model parameters, but the performance improvement gradually tends to saturate.

EfficientNet-B7 is a deep convolutional neural network architecture proposed by the Google Brain team in

2019, belonging to the EfficientNet series of models. The core innovation of this series of models is the introduction of a new model scaling method - Compound Scaling. EfficientNet-B7 is suitable for image classification tasks that require high precision and efficient computation, such as medical image analysis, satellite image processing, and complex visual recognition tasks. Its efficient architectural design allows it to perform well even in resource-constrained environments.

2.1 EfficientNet-B7 Neural Network Model Structure

EfficientNet-B7 is the largest model in the EfficientNet series, and its detailed structure is based on Compound Scaling, which achieves a balance between accuracy and computational efficiency by simultaneously adjusting the network's depth, width, and input resolution. EfficientNet-B7 is divided into three parts: the input layer, the main structure, and the output layer.

Input Layer: Input resolution: 600×600×3 . Initial convolutional layer: uses a 3×3 convolution kernel with a stride of 2, and an output channel number of 48. The role of this layer is to halve the resolution of the input image while increasing the number of channels, preparing for subsequent feature extraction.

Main Structure: The main structure of EfficientNet-B7 consists of multiple MBConv modules, which are the core building blocks of EfficientNet. Each MBConv module includes the following parts:

Expand Layer: Increases the number of channels through a 1×1 convolution kernel.

Depthwise Separable Convolution: Performs convolution operations on each channel separately to reduce computational load.

Squeeze-and-Excitation (SE) Block: Used for channel attention mechanisms to enhance feature expression.

Linear Projection Layer: Restores the number of channels to the original size through a 1×1 convolution kernel.

Residual Connection: In some modules, the input is directly added to the output to avoid the vanishing gradient problem. The improved EfficientNet-B7 connects each MBConv module through residual connections.

The main structure of EfficientNet-B7 consists of 7 main modules (Blocks), each containing multiple sub-modules (Sub-blocks), including MBConv3 and MBConv6: MBConv3 uses a 3×3 convolution kernel, while MBConv6 uses a 6×6 convolution kernel. Both types of modules include an expansion layer, depthwise separable convolution, Squeeze and Excitation (SE) block, and linear projection layer. Repeats: The number of times the MBConv module is repeated in each stage. Input Channels: The number of input channels in each stage. Output Channels: The number of output channels in each stage. Kernel Size: The size of the convolution kernel. Stride: The stride of the convolution kernel, used to control the size of the feature map. Expansion Factor: The expansion factor of the expansion layer, used to increase the number of channels. This structural design of EfficientNet-B7 allows it to perform excellently in image classification tasks while maintaining efficient computational performance.

Output Layer:Top Convolution Layer: Uses a 1×1 convolution kernel to increase the number of channels to 1280.

Global Average Pooling: Performs average global pooling on the feature map, reducing the feature map to 1×1×1280.

Fully Connected Layer: Fully connected layer, outputs the number of classes (e.g., 1000 classes for the ImageNet dataset).

Swish Activation Function: In EfficientNet, the Swish activation function is widely used, providing better gradient flow and model performance compared to the traditional ReLU activation function.

Regularization Technique - Dropout: Dropout is used before the fully connected layer to prevent overfitting.

Stochastic Depth: Randomly drops some layers during training to enhance the model's generalization ability

Training and Optimization - Optimizer: Uses the RMSProp optimizer, with a learning rate scheduler dynamically adjusting the learning rate.

Data Augmentation: Uses AutoAugment data augmentation technology to improve the model's generalization ability.

Performance - Accuracy: On the ImageNet dataset, EfficientNet-B7 achieved a top-1 accuracy of 84.3% - Number of Parameters: The number of parameters is 66M. Computational Load: The computational load is 37B FLOPS. The efficient architectural design and excellent performance of EfficientNet-B7 make it a powerful tool for image classification tasks.

As shown in Figure 1, the architecture diagram of the EfficientNet-B7 neural network model. The diagram shows the overall structure of EfficientNet-B7, including the following main parts: Stem (Root Block): The initial part of the model, used to process input images and extract preliminary features. Modules: EfficientNet-B7 consists of multiple repeated modules, each containing several sub-layers, which can be convolutional layers, batch normalization layers (Batch Normalization), activation functions, etc. The diagram shows three types of different modules (Module 1, Module 2, Module 3), which repeat in the network, and each module may contain a different number of sub-layers. Skip Connections: The "Add" operations represented by red arrows in the diagram represent skip connections, which allow the network to directly transmit information between different layers, helping to alleviate the vanishing gradient problem and promote feature reuse. Final Layers: After all modules, the network passes through a global average pooling layer (Global Average Pooling).

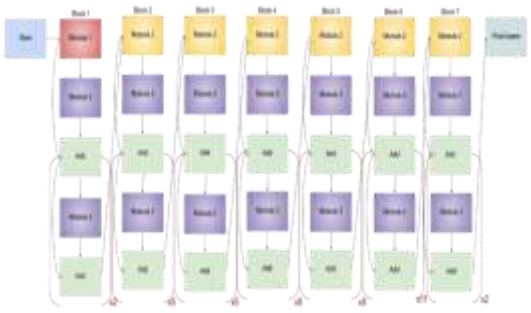

Figure 1. Architecture diagram of the EfficientNet-B7 neural network model.

2.2 Algorithm Flowchart of EfficientNet-B7 with DropPath Regularization

The main features of the EfficientNet-B7 algorithm with the introduction of the DropPath regularization technique are as follows: On the one hand, DropPath can randomly drop paths to prevent the model from over-relying on certain fixed paths, thereby avoiding overfitting. On the other hand, by forcing the network to work under different path combinations, DropPath enables the model to better adapt to different data distributions and enhances its generalization ability. For deep networks like ResNet, DropPath can effectively alleviate the vanishing gradient problem, as gradients can still propagate effectively through skip connections even if some main paths are dropped. This helps in training deeper networks. Moreover, EfficientNet-B7 uses a compound scaling method to proportionally balance depth, width, and resolution, improving accuracy while minimizing resource usage.

In summary, these features enable the EfficientNet-B7 with DropPath regularization to more effectively utilize network parameters and improve model accuracy and generalization ability when handling tasks such as image classification. The algorithm flowchart of EfficientNet-B7 with DropPath regularization :

International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

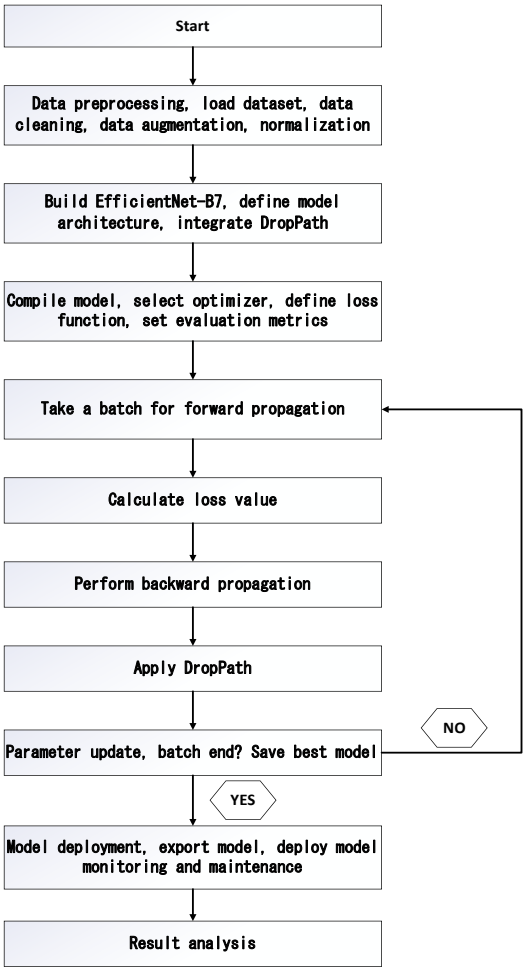

Figure 2: Algorithm Flowchart of EfficientNet-B7 with DropPath Regularization

2.2.1Forward Propagation

In neural networks, forward propagation refers to the process where data is processed through various layers of the network from the input layer to the output layer to produce the final result. For the EfficientNet-B7 model with the introduction of the DropPath regularization technique, the forward propagation process is shown in Table 2:

Table 2: Forward Propagation Process

Step	Description	Input Dimensions	Output Dimensions	Remarks
1	Input Layer	N×H×W×C	N×2H×2W×64	3x3 convol
2	Skip	Varies	Varies	output
3	Skip Connection	N×2H×2W×64	Varies	Add convolution DropPath
4	Global Average Polling	Varies	N×Cout	spatial dimensions
5	Fully Connected Layer	N×Cout	N×K	Outputs results
6	Activation Function	N×K	N×K	Softmax

Forward Propagation Steps:

Input Layer: The input data first passes through a convolutional layer, including a 3x3 convolutional layer with a stride of 2, to extract preliminary features and reduce spatial dimensions.

MBConv Module: The input data flows through multiple MBConv modules. Each MBConv module includes:Expansion Convolution: Uses 1x1 convolution to increase the number of channels.

Depthwise Convolution: Performs convolution operations separately on each input channel.

SE Module: Enhances the model's representational ability by adaptively recalibrating channel features.

Projection Convolution: Uses 1x1 convolution to reduce the number of channels, matching the feature map size with the input. In each MBConv module's residual connection, DropPath regularization is applied to discard the entire module's output with a certain probability.

Skip Connection: MBConv modules typically include skip connections, which directly add the input to the module's output.

Global Average Pooling: After all MBConv modules, the feature map passes through a global average pooling layer, reducing the spatial dimensions of each channel to a single value.

International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

Fully Connected Layer: The pooled features pass through one or more fully connected layers to ultimately output prediction results.

Activation Function: Typically, there is an activation function at the end of the network, softmax, used to convert the output into a probability distribution.

Output: Assuming the model is a classification model, the goal is to classify the input image into K categories. The final output layer's dimensions are N×K, where N is the input batch size, and K is the number of categories. Each output value represents the probability that the input image belongs to a certain category.

2.2.2 Backward Propagation

Backward propagation is a key algorithm in the training process of the improved EfficientNet-B7 neural network, used to calculate the gradients of the loss function with respect to the network parameters and use these gradients to update the network parameters. The backward propagation process is shown as follows:

Table 3. Backpropagation Process

step	Description	Remarks
1	Initialize Gradients	Prepare to start backward propagation
2	Calculate Output Layer Gradients	Use the derivative of the loss function
3	Calculate Output Layer Weights and Bias Gradients	Use the derivative of the activation function
4	Calculate Gradients Layer by Layer Backwards	Use the chain rule
5	Handle Activation Functions	Softmax activation function
6	Handle Convolutional Layers	Convolve the input feature map and the gradient of the loss with respect to the output feature map
7	Handle DropPath	Skip the dropped paths
8	Parameter Update	Adam optimizer
9	Repeat the Process	Until a predetermined number of training epochs is reached or other stopping conditions are met

Initialization of Gradients: Before starting backpropagation, the gradients of all network parameters need to be initialized to zero. This is because gradient information will be accumulated gradually during the backpropagation process.

Calculation of Output Layer Gradients: Starting from the output layer, calculate the gradient of the loss function with respect to the activation values of the output layer. This usually involves the derivative of the loss function. For example, for cross-entropy loss, it is necessary to calculate the difference between the true labels and the predicted probabilities. Then, calculate the gradient of the output layer activation values with respect to the output layer weights. This involves the derivative of the activation function. If the output layer uses the softmax activation function, then the derivative of the softmax function needs to be calculated. For an activation function f, its gradient g is:

$g=\frac{\partial f}{\partial z}$。 Here, z is the input to the activation function。

$$\text{Softmax}(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

z is a vector containing K elements, and zi is the i-th element of the vector, where K is the total number of classes. The softmax function ensures that the K output values are all positive and their sum is 1.

Layer-by-Layer Gradient Calculation: Calculate gradients from the output layer to the previous layers layer by layer. For each layer, calculate the gradient of the loss with respect to the activation values of that layer, and then calculate the gradient of the activation values with respect to the weights and biases of that layer. This process involves the chain rule, which means calculating the gradient of the previous layer based on the known gradients.

Handling Activation Functions and Convolutional Layers: For the softmax activation function, calculate its derivative. For convolutional layers, calculate the gradient of the convolutional kernel, which involves performing a convolution operation on the input feature map and the gradient of the loss with respect to the output feature map. The gradient of the weights w for a convolutional layer involves the following convolution operation:

$$\frac{\partial \mathrm{L}}{\partial \mathrm{W}} = \sum_{\mathrm{x,y}} \frac{\partial \mathrm{L}}{\partial \hat{\mathrm{y}}(\mathrm{x,y})} \cdot \frac{\partial \hat{\mathrm{y}}(\mathrm{x,y})}{\partial \mathrm{W}}$$

where x,y represent the position of the output feature map.

Special Case of Applying DropPath: At locations where DropPath is introduced, since some paths are randomly dropped during the forward propagation, the gradients of these paths should be zero during backpropagation. This means that when calculating gradients, these dropped paths need to be skipped. D represents the output of the DropPath layer, and its gradient is:

$$\frac{\partial \mathrm{L}}{\partial \mathrm{D}} = \begin{cases} \frac{\partial \mathrm{L}}{\partial \mathrm{D}} & \text{if } \mathrm{D} \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

Parameter Update: Once the gradients of all parameters have been calculated, the network parameters can be updated using the Adam optimizer. The optimizer adjusts the parameters based on the gradients and learning rate to minimize the value of the loss function.

Repeat the Process: Repeat the above process in each training epoch until the predetermined number of training epochs is reached or other stopping conditions are met. In practical applications, backpropagation is usually combined with various optimization algorithms and regularization techniques to improve the accuracy and generalization ability of the model.

3. Flower Dataset

Google Cloud provides a commonly used flower dataset with 104 species, stored in .tfrec binary format. The dataset is available in four different sizes: 192x192, 224x224, 331x331, and 512x512.As shown in Table 4, the flower dataset in different sizes is listed.

Table 4. Flower Dataset

Size	Set	Count	File Format
192x192	Test Val Train	0-15	.tfrec
224x224	Test Val Train	0-15	.tfrec
331x331	Test Val Train	0-15	.tfrec
512x512	Test Val Train	0-15	.tfrec

4. Flower Classification Based on EfficientNet-B7 Model

Flower classification based on the EfficientNet-B7 model is approached from four aspects. The first step is to define the dataset, showing the sizes of the training, validation, and testing datasets, and randomly sampling to display images from the dataset. The second step involves setting up the EfficientNet-B7 model in preparation for model training. The third step trains the EfficientNet-B7 model using flower training samples to achieve flower classification and analyzes the results.

The final step evaluates the model, assessing its performance using F1 score, precision, and recall.

4.1 Defining the Dataset

The commonly used flower dataset, comprising 104 species, is available on Google Cloud in the form of `.tfrec` binary storage files. For this experiment, the 512x512 size dataset is selected for classification validation. The dataset is divided into training, validation, and testing sets in the ratio of 70%, 15%, and 15%, respectively. Image pixel values are normalized to the range [0,1] to accelerate model training. To enhance the model's generalization capability, data augmentation can be applied to the training data.

4.2 Setting Up the EfficientNet-B7 Model

Model Architecture Selection: EfficientNet-B7 is one of the largest models in the EfficientNet series, offering high performance and computational efficiency. It optimizes the network's depth, width, and input resolution through compound scaling techniques, making it suitable for complex image classification tasks.

Loading Pre-trained Weights: EfficientNet-B7 typically comes with pre-trained weights on the ImageNet dataset. These weights can serve as initial model parameters, facilitating faster convergence and improved performance on new tasks. When loading pre-trained weights, one can choose whether to include the model's top (i.e., fully connected layer).

Freezing Convolutional Layers: To leverage the pre-trained model's feature extraction capabilities while reducing training time and computational resources, the convolutional layers of the pre-trained model are usually frozen. Freezing means that the weights of these layers will not be updated during training. With frozen convolutional layers, the model can focus on learning the new classification task without relearning basic image features.

Modifying the Model Top : The original top of the

model is designed for the ImageNet dataset and typically includes a global average pooling layer and a fully connected layer with 1000 outputs (the number of ImageNet classes). To adapt the model to a new classification task (such as flower classification), the model top needs to be modified as follows:

1. Global Average Pooling Layer: Converts feature maps into fixed-size feature vectors for input to the fully connected layer.

2. Dropout Layer: Adds a Dropout layer to reduce overfitting. During training, the Dropout layer randomly discards some neurons, enhancing the model's generalization ability.

3. Custom Fully Connected Layer: Adds a new fully connected layer based on the number of classes in the new task (e.g., 104 classes for flower classification) and uses the softmax activation function to output classification probabilities.

Compiling the Model: After defining the model structure, it needs to be compiled, specifying the optimizer, loss function, and evaluation metrics:

1. Optimizer: Choose a suitable optimizer, such as the Adam optimizer, to update the model's weights.

2. Loss Function: For multi-class classification tasks, the categorical cross-entropy loss function is commonly used.

3. Evaluation Metrics: Accuracy is typically used as the metric to evaluate model performance.

Model Summary: To better understand the model's structure and number of parameters, the model summary can be printed. The summary includes the name, output shape, and number of parameters for each layer, as well as the total number of parameters, trainable parameters, and non-trainable parameters in the model. Table 5 below shows the structure of the EfficientNet-B7model.

International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

Table 5. EfficientNet-B7 Model Architecture

Layer (type)	Output Shape	Param
lambda	(None, 224, 224, 3)	0
keras_layer	(None, 2560)	64097680
dense	(None, 3)	7683
Total params: 64,105,363		
Trainable params: 63,794,643		
Non-trainable params: 310,720		

Model Architecture:

1. Lambda Layer (lambda_1) : Output Shape:(None, 512, 512, 3),Number of Parameters: 0,Description: The Lambda layer is typically used to implement custom operations. The output shape (None, 512, 512, 3)indicates that the layer produces a 4-dimensional tensor. Here, `None` represents the batch size, which can be dynamically adjusted as needed. `512, 512` indicates the height and width of the image, meaning the input image is resized to 512×512 pixels. The `3` represents the number of channels (for RGB images). The parameter count is 0, indicating that this layer has no trainable parameters.

2. KerasLayer (keras_layer_1):Output Shape: (None, 2560),Number of Parameters: 64,097,680,Description: KerasLayer is a layer that encapsulates a pre-trained model and is part of EfficientNet-B7. The output shape `(None, 2560)` indicates that the layer produces a 2-dimensional tensor. `None` represents the batch size, and `2560` represents the feature dimension of the output, which is the number of features from the output layer of the EfficientNet-B7 model. The total number of parameters in this layer is 64,097,680, including both trainable and non-trainable parameters.

3. Dense Layer (dense_1):Output Shape: (None, 104), Number of Parameters: 266,344, Description: The Dense layer is a fully connected layer commonly used to map features to target classes. The output shape `(None, 104)` indicates that the layer produces a 2-dimensional tensor. `None` represents the batch size, and `104` represents the number of output classes, indicating that this is a 104-class classification task. The total number of parameters in this layer is 266,344.

Summary of Model Parameters:Total Parameters are 64,364,024.This is the total number of parameters in the model, including both trainable and non-trainable parameters.Trainable Parameters are 64,053,304 These are the parameters that can be updated during training, typically including weights and biases.Non trainable Parameters are 310,720These are fixed parameters in the model, usually from the pre-trained model weights. They do not update during training.Summary of Model Architecture:Input Layer: The input image is resized to a 512×512 RGB image.Pre-trained Model Layer: A pre-trained model (such as EfficientNet-B7) is used, with an output feature dimension of 2560.Fully Connected Layer: The features from the pre-trained model are mapped to 104 classes.

This architectural design fully leverages the powerful feature extraction capabilities of the pre-trained model and applies them to a specific classification task through the fully connected layer.

4.3 Training the EfficientNet-B7 Model

Model Training:

1. Initiating Training:The model.fit() method is used to start model training. It specifies the training dataset, the number of steps per epoch, the total number of training epochs, the validation dataset, the number of validation steps, and the callback functions.

2. Dynamic Learning Rate Adjustment:At the beginning of each epoch, the `LearningRateScheduler` callback function calls the `lrfn` function to update the learning rate. In this experiment:

LR_START: The initial learning rate is set to 0.00001.

LR_MIN: The minimum learning rate is set to 0.00001.

LR_MAX: The maximum learning rate is set to 0.00005×`strategy.num_replicas_in_sync`.`strategy.num_replicas_in_sync` represents the number of replicas in distributed training and is used to adjust the learning rate based on the number of devices.

LR_RAMPUP_EPOCHS: The number of epochs for learning rate ramp-up is set to 4. During these epochs,

International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

the learning rate linearly increases from `LR_START` to `LR_MAX`.

LR_SUSTAIN_EPOCHS: The number of epochs to sustain the learning rate is set to 0. During these epochs, the learning rate remains at `LR_MAX`.

LR_EXP_DECAY: The exponential decay factor for the learning rate is set to 0.8. After the ramp-up and sustain phases, the learning rate will decay exponentially.

The following figure illustrates a learning rate schedule that changes over time or training epochs.

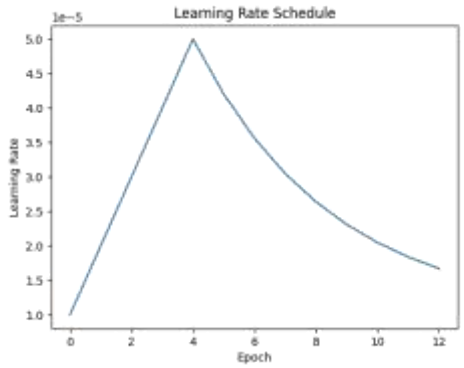

Figure 3. Learning Rate Schedule

The chart shows three phases of change in the learning rate:

Linear Increase Phase: At the beginning of training, the learning rate starts from a small initial value (the left endpoint in the figure). As training progresses, the learning rate increases linearly until it reaches a set maximum value (the peak in the figure). This phase usually lasts for several epochs, with the goal of allowing the model to quickly escape local minima in the early stages of training.

Stable Phase: Once the learning rate reaches the maximum value, it remains at this level for a period of time (the horizontal line segment in the figure). During this phase, the model continues to train using the higher learning rate to further optimize the weights.

Exponential Decay Phase: After reaching the maximum value and maintaining it for a while, the learning rate begins to decay exponentially (the downward curve in the figure). This means that the learning rate gradually decreases with each epoch, typically reduced by a certain decay factor (e.g., 0.8). The decay phase helps the model make finer weight adjustments in the later stages of training, avoiding large steps that could cause it to overshoot the optimal solution.

This learning rate schedule strategy helps the model to adopt different learning strategies at different stages of training, thereby improving training efficiency and model performance.

3. Monitoring the Training Process:After each epoch, print the training loss , training accuracy , validation loss and validation accuracy .

4. Training Completion:After 13 training epochs, the model's training loss gradually decreases, and the training accuracy gradually increases. The validation loss and validation accuracy also show similar trends, indicating that the model's performance on the validation set is gradually improving. Figures 4 and 5 show the changes in loss and accuracy during the training process of the improved model.

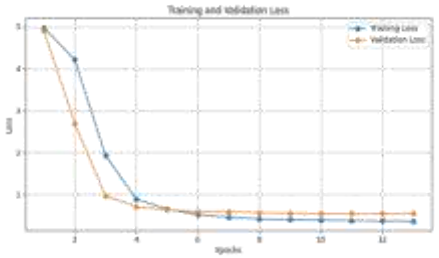

Figure 4. Loss Values of the EfficientNet-B7 Model

It can be seen from this figure that the model performs well during the training process, successfully reducing the loss. The introduction of the DropPath regularization technique in the improved model has achieved good fitting effects on both the training and validation data.

International Journal of Science and Engineering Applications
Volume 14-Issue 04, 42 – 52, 2025, ISSN:- 2319 - 7560
DOI: 10.7753/IJSEA1404.1007

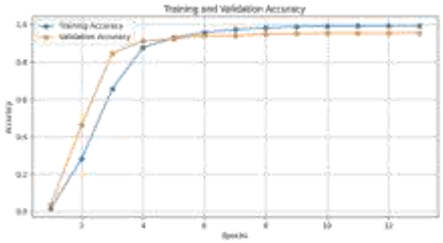

Figure 5. Accuracy of the EfficientNet-B7 Model

It can be seen from this figure that the model performs well during the training process, successfully increasing the accuracy and achieving good classification results on both the training and validation data. However, the specific performance still needs to be further evaluated through other metrics such as precision, recall, and F1 score.

4.4 Performance Evaluation of the EfficientNet-B7 Model

As shown in Figure 6, the confusion matrix evaluates the performance of the EfficientNet-B7 model using three metrics: precision, recall, and F1 score.

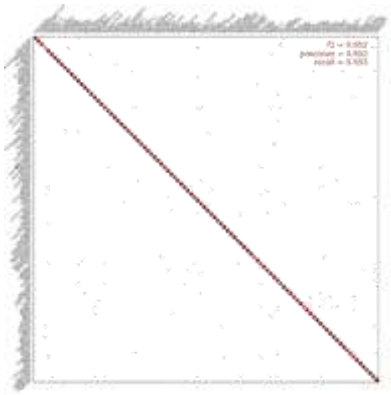

Figure 6. Confusion Matrix

In the figure, the rows represent the actual class labels, while the columns represent the predicted class labels by the model. As can be seen from the confusion matrix, the model achieves an F1 score of 0.952, precision of 0.952, and recall of 0.955 in the classification task, demonstrating its excellent performance in classification.

5 Conclusion

This paper takes flowers as the research subject and proposes an improved flower classification model using the EfficientNet-B7 convolutional neural network with the incorporation of the DropPath regularization term, based on the flower dataset provided on Google Cloud, which includes 104 species. The model reduces the training time and effectively solves the classification task of iris flowers. It can be widely applied to the recognition of other plant species.

6.References

[1] Fröhlich, B. and Plate, J. (2000). The cubic mouse: a new device for three-dimensional input. In Proceedings of the SIGCHI Conference on Human Factors in Computing Systems.

[2] Guru, D.S., Sharath Kumar, Y.H., Manjunath, S. (2011). Textural features in flower classification. Math. Comput. Modell. 54(3–4), 1030–1036.

[3] Mabrouk, A., Najjar, A., Zagrouba, E. (2014). Image flower recognition based on a new method for color feature extraction. In: VISAPP 2014 - Proceedings of the 9th International Conference on Computer Vision Theory and Applications, vol. 2.

[4] Huang, G., Liu, Z., Van Der Maaten, L.(2017). Densely connected convolutional networks. In: 2017 Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition. IEEE.

[5] Shaparia, R., Patel, N., Shah, Z. (2017). Flower classification using texture and color features.

[6] Antonelli, A., et al.(2020). State of the world's plants and fungi 2020. Royal Botanic Gardens, Kew.

[7] Narvekar, C., Rao, M. (2020). Flower classification using CNN and transfer learning in CNN-agriculture perspective. In: 2020 3rd International Conference on Intelligent Sustainable Systems (ICISS), pp. 660–664.

[8] Alipour, N., Tarkhaneh, O., Awrangjeb, M., Tian, H.(2021). Flower image classification using deep convolutional neural network. Pp. 1–4.

[9] Tan, M., Le, Q. (2019). EfficientNet: rethinking model scaling for convolutional neural networks. arXiv:1905.11946v5.

[10] Alipour, N., Tarkhaneh, O., Awrangjeb, M., Tian, H. (2021). Flower image classification using deep convolutional neural network. Pp. 1–4.