Cocoa Diseases Classification using Deep Learning Algorithm

Ker Sing Soh[1], *Ervin* Gubin Moung[1], John Julius Danker Khoo[1], Jamal Ahmad Dargham[2], and *Ali* Farzamnia[2]

[1]Faculty of Computing and Informatics, Universiti Malaysia Sabah, Kota Kinabalu, Malaysia
[2]Faculty of Engineering, Universiti Malaysia Sabah, Kota Kinabalu, Malaysia

Abstract. This work addresses the critical issue of cocoa plant diseases, which pose a significant threat to global agriculture and the livelihood of millions of farmers. Convolutional Neural Networks (CNN) has been utilized for classifying cocoa diseases, focusing on combating the significant agricultural and economic impacts of black pod rot and pod borer. Utilizing a dataset of 4390 images, five CNN architectures—Custom CNN, VGG-16, EfficientNetB0, ResNet50, and LeNet-5—were assessed for their ability to accurately identify disease presence. The Custom CNN model was found to be the most effective, achieving an accuracy of 91.79%, precision of 91.79%, recall of 91.79%, F1 score of 82.08%, sensitivity of 96.69%, and specificity of 98.40%, indicating its strong capability in correctly classifying both healthy and diseased plants. The work methodically approached data preprocessing, model parameter tuning, and a structured machine learning evaluation process. While EfficientNetB0 and ResNet50 also displayed commendable performances, VGG-16 and LeNet-5 lagged, suggesting the need for further model refinement. This work underscores the importance of the F1 score in evaluating model performance, especially given the class imbalance within the dataset. The findings suggest future exploration into other pre-trained models and data augmentation strategies to further enhance classification accuracy. The goal of this work is the implementation of a real-time system to minimize cocoa production losses due to disease.

1 Introduction

Cocoa refers to a famous tropical fruit that is also a primary ingredient in chocolate production. It is considered a main source of income for millions of farmers worldwide. However, sudden onset diseases significantly threaten this livelihood, including the black pod rot disease and pod borer, known scientifically as Phytophthora pod rot and Conopomorpha cramerella, respectively. These common diseases affect both the internal and external surfaces of the fruit. They are primarily caused by pests, resulting in a decrease in both the cocoa cultivation area and agricultural productivity [1]. Furthermore, these diseases disrupt the developmental phases of the crop, making it difficult for farmers to harvest. The increasing incidence of plant diseases has escalated into a significant concern within the sphere of agriculture, necessitating the exploration of innovative diagnostic solutions. The evolution of computational sciences, particularly over the last several decades, has heralded

the advent of machine learning — an extension of artificial intelligence that augments the analytical capabilities of computers. This sophisticated toolset is adept at navigating intricate operations, notably in the realms of image classification [2, 3], pattern recognition [4, 5], anomaly detection [6, 7], and pictorial segmentation [8]. Among the plethora of methodologies, Convolutional Neural Networks (CNNs) stand out for their proficiency in handling expansive datasets beyond the scope of traditional algorithmic strategies, thereby yielding enhanced precision in a variety of analytical tasks. In the agricultural context, deep learning paradigms, encapsulated by CNNs, have been instrumental in surmounting the obstacles encountered in crop cultivation. An illustrative investigation employing CNNs for the categorization of cacao pods [9] underscored the efficacy of the EfficientNetB0 archetype, positing it as the optimal model for said task. The investigation also underlined the imperative for continued scholarly inquiry in this avenue.

In pursuit of resolving the classification conundrum, the development of a tailored CNN architecture, comprised of several stratified layers, is proposed. In addition, pre-trained models, also known as CNNs, are frequently utilized to enhance performance metrics. Research studies employing various CNN algorithms have successfully classified image data, yielding good results [10–13]. CNNs are particularly adept at automatically extracting features from data. Specifically, these models will be constructed and subsequently trained using the provided images.

2 Methods

2.1 Dataset preparation and model parameters initialization

In this study, the dataset, known as the Cacao Disease Dataset, was curated from the Kaggle website and manually downloaded. It comprises 4390 images, each with a resolution of 1080×1080, categorized into three classes: Healthy cocoa, Black pod rot, and Pod Borer, as detailed in Table 1. The dataset was divided into training, validation, and testing sets in a 7:2:1 ratio, with a predominance of images from the healthy class.

Table 1. Data split for each classes

Classes	Training dataset	Validation dataset	Testing dataset	Total
Healthy	2340	669	335	3344
Black Pod Rot	660	188	95	943
Pod Borer	72	20	11	103

The entire dataset consists of images in .jpg format, which are utilized in this research study, as illustrated in Figs. 1, 2, and 3. To maintain consistency in image data and quality, each image will be resized to 60×60 pixels and the color space converted to RGB.

Fig. 1. Sample of Cocoa Diseases Dataset from healthy class.

Fig. 2. Sample of Cocoa Diseases Dataset from unhealthy class (Black Pod Rot Diseases).

(c)

Fig. 3. Sample of Cocoa Diseases Dataset from unhealthy class (Pod Borer Diseases).

Before model initialization, parameters and optimizations are established. A batch size of 64 is chosen, as larger sizes could impede the network's generalization—a size of 32-64 is optimal for effective network updates. Additionally, the learning rate is set at 0.1 for 100

epochs, employing Adamax optimization [14–17]. This approach allows for an automatic reduction in the learning rate as weight adjustments become less variable. Adamax, an Adam variant, replaces the optimizer's second-order moment with an infinity norm, thus enhancing stability and enabling automatic adaptation of the learning rate for each parameter in the optimization process. Table 2 presents the standard training parameters for all proposed CNN-based architectures.

Table 2. Standard training parameters for CNN-based architectures

Parameter	Value	Details
Batch Size	64	Optimal for balancing the update frequency and generalization capability of the network.
Learning Rate	0.1	Initial value, with automatic reduction to manage weight variability during training.
Epochs	100	Total number of complete passes through the training dataset.
Optimization Method	Adamax	A variant of the Adam optimizer, utilizing the infinity norm for more stable updates.

2.2 Custom Convolutional Neural Network (Custom CNN)

CNN is a specialized neural network for processing data with a grid-like topology, such as images. This CNN begins with a convolutional block (Conv Block 1), which applies 64 (3×3) filters with the 'same' padding, followed by ReLU activation, 2×2 max pooling, and dropout for regularization. Batch normalization is applied after each convolutional layer to stabilize learning and accelerate training. Subsequent convolutional blocks increase the filter size from 128 (5×5) in Conv Block 2, to 256 (3×3) in Conv Block 3, and finally to 512 (3×3) in Conv Block 4, each following the same pattern of ReLU activation, max pooling, dropout, and batch normalization. After the convolutional blocks, the network flattens the feature maps to prepare them for fully connected layers. The first fully connected block has 256 neurons, and the second has 512 neurons, both using ReLU activation and dropout. Batch normalization follows each dense layer to maintain the benefits throughout the network. The final layer of the network is a dense layer with 3 neurons corresponding to the number of classes, employing a softmax activation function to output probabilities for each class. This structure allows the CNN to classify input data into three categories effectively. Table 3 shows the details of Custom CNN architecture used in this work.

Table 3. Custom CNN architecture

Layer	Operator & Details	Activation & Regularization	Notes
1	Conv2D 64 (3×3, padding=same)	ReLU, MaxPooling (2×2), Dropout (0.25)	Conv Block 1
2	BatchNormalization	-	After Conv2D layers
3	Conv2D 128 (5×5, padding=same)	ReLU, MaxPooling (2×2), Dropout (0.25)	Conv Block 2
4	BatchNormalization	-	After Conv2D layers
5	Conv2D 256 (3×3, padding=same)	ReLU, MaxPooling (2×2), Dropout (0.25)	Conv Block 3
6	BatchNormalization	-	After Conv2D layers
7	Conv2D 512 (3×3, padding=same)	ReLU, MaxPooling (2×2), Dropout (0.25)	Conv Block 4
8	BatchNormalization	-	After Conv2D layers
9	Flatten	-	Prepares for FC layers

10	Dense 256	ReLU, Dropout (0.25)	Fully connected Block 1
11	BatchNormalization	-	After Dense layers
12	Dense 512	ReLU, Dropout (0.25)	Fully connected Block 2
13	BatchNormalization	-	After Dense layers
14	Dense 3 (activation='softmax')	-	Output layer, 3 classes

2.3 Visual Geometry Group (VGG16)

VGG, a deep CNN, is trained using the ImageNet dataset and consists of 16 convolutional layers and 4 pooling layers, followed by 3 fully connected layers. The network starts with an input of 224×224 RGB images. The first and second layers are convolutional with 64 filters of size 3×3, using a stride of 1 and 'same' padding, maintaining the output dimension at 224×224×64. These are followed by a max pooling layer with a 2×2 window and a stride of 2, which halves the output dimensions. Layers four and five are convolutional, with 128 filters, again with a stride of 1 and 'same' padding, resulting in an output of 112×112×128. Another max pooling layer follows, reducing dimensions by half. Layers seven and eight consist of 256 filters, keeping the output at 56×56×256. This is followed by max pooling, halving the dimensions once more. The next set, layers 10 to 12, involve 512 filters and maintain an output dimension of 28×28×512, followed by max pooling. Layers 14 and 15, also with 512 filters, maintain the output at 14×14×512, followed by a final max pooling layer that reduces the size to 7×7×512. After the convolutional and pooling layers, the network flattens the output to feed into the dense layers. The first two dense layers each have 4096 units, and the final layer has 3 units corresponding to the number of classes, with a softmax activation function to classify the input into one of three categories. Table 4 shows the details of VGG16 architecture used in this work.

Table 4. VGG16 architecture

Layer	Operator & Details	Activation & Regularization	Notes
1-2	Conv2D 64 (3×3, stride=1, padding=same)	ReLU	Input 224×224 RGB, output 224×224x64
3	MaxPooling (2×2, stride=2)	-	Output dimensions halved
4-5	Conv2D 128 (3×3, stride=1, padding=same)	ReLU	Output dimensions remain 112×112×128
6	MaxPooling (2×2, stride=2)	-	Output dimensions halved
7-8	Conv2D 256 (3×3, stride=1, padding=same)	ReLU	Output dimensions remain 56×56×256
9	MaxPooling (2×2, stride=2)	-	Output dimensions halved
10-12	Conv2D 512 (3×3, stride=1, padding=same)	ReLU	Two sets, output 28×28×512
13	MaxPooling (2×2, stride=2)	-	Output dimensions halved
14-15	Conv2D 512 (3×3, stride=1, padding=same)	ReLU	Output remains 14×14×512
16	MaxPooling (2×2, stride=2)	-	Final output dimensions 7×7×512
17	Flatten	-	Flatten for dense layers
18	Dense 4096	ReLU	Fully connected dense layer
19	Dense 4096	ReLU	Second fully connected dense layer
20	Dense 3	Softmax	Output layer for 3 classes

2.4 EfficientNetB0

EfficientNetB0 is a sophisticated neural network architecture within the realm of deep learning. It is composed of a sequence of layers specifically designed to balance accuracy and computational efficiency. The first layer is a convolutional layer with a 3×3 kernel, Swish activation, and batch normalization, designed to process input images of size 224 × 224 with 32 channels. In the construction of the neural network, the second tier, designated as MBConv1 and characterized by a 3×3 kernel dimension, orchestrates a series of computational processes. Initially, it engages a 1×1 convolutional process, which segues into a depthwise convolution. This is succeeded by the integration of squeeze-and-excitation (SE) blocks, a sophisticated mechanism that enables the model to recalibrate the channel-wise feature responses by explicitly modelling interdependencies between channels. A concluding 1×1 convolution is applied, subsequently followed by the deployment of a Swish activation function and a batch normalization procedure. This intricate sequence operates to condense the image resolution to 112 × 112 while steadfastly preserving the 16-channel architecture, thereby eschewing any channel expansion. Progressing further, the architecture adopts MBConv6 blocks, which are incorporated repetitively with discernible variations in kernel dimensions and expansion ratios. The third stratum of the network diligently replicates this block twice, utilizing a 3×3 kernel that achieves a sextuple expansion and augments the channel count to 24, maintaining the preceding resolution. The subsequent fourth stratum also perpetuates the MBConv6 block duplication; however, it distinguishes itself by employing a 5×5 kernel, which serves to refine the resolution to 56 × 56 and escalate the channel density to 40. The fifth layer has three repetitions of the MBConv6 block with a 3×3 kernel, yielding 80 channels at a 28 × 28 resolution. The sixth layer follows the same pattern but with a 5×5 kernel, producing 112 channels at a resolution of 14 × 14, and this configuration is also used in the seventh layer, which features four repetitions and increases the channels to 192. The eighth layer consists of an MBConv6 block with a 3×3 kernel, expanding the channels to 320 at a 7 × 7 resolution. The final layer combines a 1×1 convolution, global average pooling, and a dense layer with three units for classification, employing Swish activation for the convolution and Softmax for the dense layer, accommodating the architecture to classify three classes with a final output of 1280 channels. Table 5 shows the details of EfficientNetB0 architecture used in this work.

Table 5. EfficientNetB0 architecture

Layer	Operator & Details	Activation & Regularization	Notes
1	Conv3×3	Swish, BatchNormalization	Input: 224 × 224, 32 channels
2	MBConv1, k3×3 (1×1 conv, depthwise conv, SE, 1×1 conv)	Swish, BatchNormalization	Resolution: 112 × 112, 16 channels, No expansion
3	MBConv6, k3×3 (1×1 conv, depthwise conv, SE, 1×1 conv) ×2	Swish, BatchNormalization	Resolution: 112 × 112, 24 channels, 6x expansion, Repeat block x2
4	MBConv6, k5×5 (1x1 conv, depthwise conv, SE, 1×1 conv) ×2	Swish, BatchNormalization	Resolution: 56 × 56, 40 channels, 6x expansion, Repeat block x2
5	MBConv6, k3×3 (1×1 conv, depthwise conv, SE, 1×1 conv) ×3	Swish, BatchNormalization	Resolution: 28 × 28, 80 channels, 6x expansion, Repeat block x3

6	MBConv6, k5×5 (1×1 conv, depthwise conv, SE, 1×1 conv) ×3	Swish, BatchNormalization	Resolution: 14 × 14, 112 channels, 6x expansion, Repeat block x3
7	MBConv6, k5×5 (1×1 conv, depthwise conv, SE, 1×1 conv) ×4	Swish, BatchNormalization	Resolution: 14 × 14, 192 channels, 6x expansion, Repeat block x4
8	MBConv6, k3×3 (1×1 conv, depthwise conv, SE, 1×1 conv)	Swish, BatchNormalization	Resolution: 7 × 7, 320 channels, 6x expansion
9	Conv1×1 & GlobalAveragePooling2D & Dense 3	Swish (Conv1×1), Softmax (Dense)	Final layer with 1280 channels, tailored for 3 classes

2.5 Deep Residual Network (ResNet50)

ResNet, known for its "identity shortcut connection," enables the model to skip over certain layers. ResNet50, a 50-layer-deep CNN, is pretrained, having been previously trained on a substantial dataset. It is structured into 5 stages, each consisting of a convolution block and an identity block, both comprising three convolutional layers. The initial input image is processed by a convolution layer with a kernel size of 7×7, employing 64 different kernels and a stride of size 2, followed by a max pooling layer with the same stride size. Subsequently, in the convolution blocks, a sequence of 1×1 convolution with 64 kernels, 3×3 convolution with 64 kernels, and finally, 1×1 convolution with 256 kernels is used. This sequence is repeated thrice, resulting in 9 layers for that stage. In the succeeding stage, the sequence of 1×1 kernel with size 128, 3×3 kernel with size 128, and 1×1 kernel with size 512 is repeated four times, creating 12 layers. Then, the configuration of 1×1, 3×3, and 1×1 kernels with sizes of 256, 256, and 1024, respectively, is replicated six times, culminating in 18 layers. Furthermore, the arrangement of 1×1 with size 512, 3×3 with size 512, and 1×1 with size 2048 kernels is repeated three times, contributing another 9 layers. The architecture concludes with an average pooling operation and a fully connected layer incorporating a softmax function, amounting to one layer. The total count of layers thus sums to 50. Table 6 shows the details of ResNet50 architecture used in this work.

Table 6. ResNet50 architecture

Layer	Operator & Details	Activation & Regularization	Notes
1	Conv2D (7×7), Stride (2,2), Padding 'Valid'	ReLU, BatchNormalization	Initial convolution, Output: 112×112, 64 channels
2	MaxPooling2D (3×3), Stride (2,2), Padding 'Valid'	-	Downsample, Output: 56×56
3-4	Conv Block: 1×1 Conv, 3×3 Conv, 1×1 Conv (64, 64, 256 channels)	ReLU, BatchNormalization	Output: 56×56, 256 channels, 3 layers
5-7	Identity Block ×3: 1×1 Conv, 3×3 Conv, 1×1 Conv (64, 64, 256 channels)	ReLU, BatchNormalization	Output: 56×56, 256 channels, 9 layers (3 layers each ×3)
8	Conv Block: 1×1 Conv, 3×3 Conv, 1×1 Conv (128, 128, 512 channels)	ReLU, BatchNormalization	Output: 28×28, 512 channels, 3 layers
9-14	Identity Block ×3: 1×1 Conv, 3×3 Conv, 1×1 Conv (128, 128, 512 channels)	ReLU, BatchNormalization	Output: 28×28, 512 channels, 9 layers (3 layers each ×3)

15	Conv Block: 1×1 Conv, 3×3 Conv, 1×1 Conv (256, 256, 1024 channels)	ReLU, BatchNormalization	Output: 14×14, 1024 channels, 3 layers
16-29	Identity Block ×6: 1×1 Conv, 3×3 Conv, 1×1 Conv (256, 256, 1024 channels)	ReLU, BatchNormalization	Output: 14×14, 1024 channels, 18 layers (3 layers each ×6)
30	Conv Block: 1×1 Conv, 3×3 Conv, 1×1 Conv (512, 512, 2048 channels)	ReLU, BatchNormalization	Output: 7×7, 2048 channels, 3 layers
31-49	Identity Block ×3: 1×1 Conv, 3×3 Conv, 1×1 Conv (512, 512, 2048 channels)	ReLU, BatchNormalization	Output: 7×7, 2048 channels, 9 layers (3 layers each ×3)
50	AveragePooling2D followed by a Fully Connected layer with 3 units	Softmax	Global average pooling, Dense layer for 3 classes

2.6 LeNet-5

The LeNet-5 architecture, a foundational model in CNN architectures, consists of seven distinct layers. The first layer is a convolutional layer with 6 filters, a kernel size of 5×5, a stride of 1, and 'valid' padding, which focuses on capturing basic features and produces six feature maps. The second layer is an average pooling layer with a pool size of 2×2 and a stride of 2, reducing the spatial resolution without learning parameters. The third layer is another convolutional layer, this time with 16 filters, which extracts more complex features and yields 16 feature maps. This is followed by another average pooling layer with the same specifications as the second layer, further reducing spatial resolution. The fifth layer is a dense fully connected layer with 120 units, where the classification process begins. The sixth layer is a continuation of this process, a dense layer with 84 units. The architecture's design is finalized by a dense layer, consisting of three units that correspond to the class categories. This layer applies a softmax activation to calculate class probabilities, effectively translating the network's outputs into a probabilistic assessment aligned with the number of classes.

Table 7. LeNet-5 architecture

Layer	Operator & Details	Activation & Regularization	Notes
1	Conv2D: 6 filters, kernel size (5,5), stride (1,1), padding 'valid'	ReLU	Captures basic features, produces 6 feature maps
2	AveragePooling2D: pool size (2,2), stride (2,2)	-	Reduces spatial resolution, not a learnable layer
3	Conv2D: 16 filters, kernel size (5,5), stride (1,1), padding 'valid'	ReLU	Extracts more complex features, produces 16 feature maps
4	AveragePooling2D: pool size (2,2), stride (2,2)	-	Further reduces spatial resolution, not a learnable layer
5	Dense: 120 units	ReLU	Fully connected layer, begins classification process
6	Dense: 84 units	ReLU	Fully connected layer, continues classification process
7	Dense: 3 units (number of classes), activation='softmax'	Softmax	Output layer, produces probabilities for 3 classes

2.7 Model Training

The work incorporates training, validation, and testing datasets to measure the effectiveness of the deep learning models proposed earlier. The models are applied to these datasets to facilitate a comparative analysis, a process illustrated in Fig. 4's flowchart. This flowchart outlines a structured methodology for the development of a machine learning model, encompassing each stage from the initial handling of data to the reporting of results. The process begins with data preparation, involving the assignment of class labels. Following this is data preprocessing, where the data is encoded and transformed within the RGB colour space. The model building phase is marked by the initialization of the architecture and its compilation with an appropriate optimizer and loss function. The subsequent training phase fits the model to the training data and validates it against a separate dataset. The evaluation and visualization phase allows for the assessment of performance metrics like accuracy and loss. Here, results, including model configurations, may be saved. The workflow then advances to the prediction phase, using the trained model on a test set, followed by an accuracy calculation step. The flowchart concludes in the reporting of results, providing a summarization of the machine learning pipeline's outcomes.

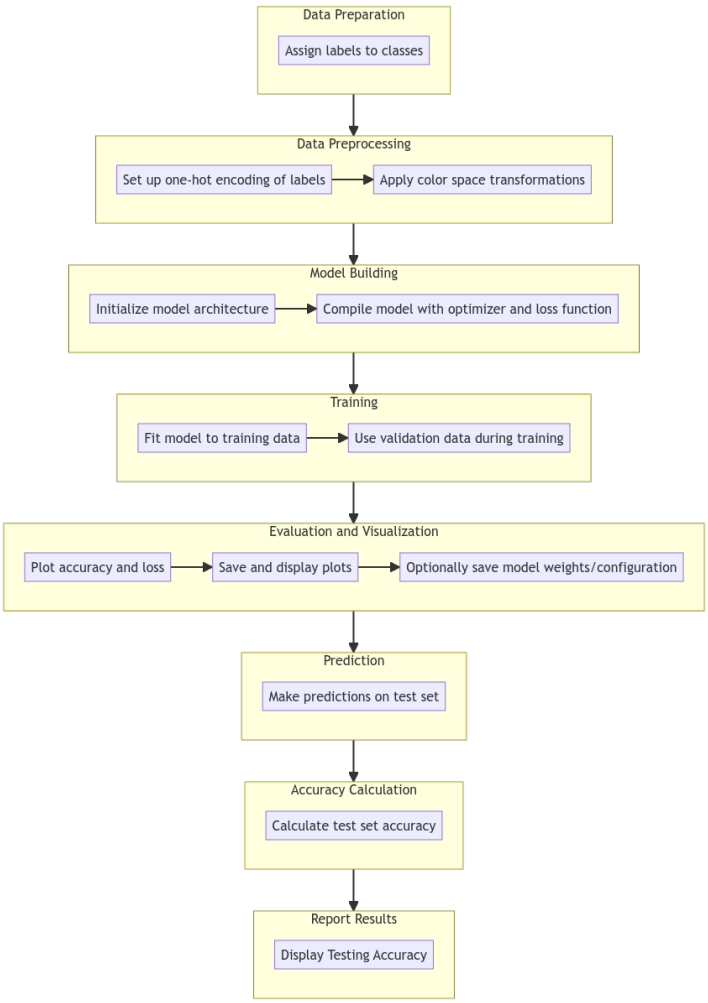

Fig. 4. The proposed flowchart for this work.

2.8 Performance Measures

In the model evaluation section, the metrics used to estimate the model's predictive abilities are covered. Specifically, performance measures such as Accuracy, Precision, Recall, F1 Score, Sensitivity, and Specificity are computed because they are pertinent to the dataset and the classification tasks. These parameters are associated with True Positives and False Positives (TP/FP) — the counts of correctly and incorrectly predicted positive instances, respectively — and True Negatives and False Negatives (TN/FN) — the counts of correctly and incorrectly predicted negative instances, respectively. Accuracy is the proportion of correctly predicted observations out of the total observations and is calculated as shown in (1).

$$Accuracy = \frac{True\ Positives + True\ Negatives}{Total\ Observations} \tag{1}$$

Precision is the ratio of correctly predicted positive observations to the total predicted positive observations. It is calculated as in (2).

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \tag{2}$$

Recall is the ratio of correctly predicted positive observations to all observations in actual class. It is calculated as in (3).

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{3}$$

F1 Score is the harmonic mean of precision and recall. It is calculated as in (4).

$$F1\ Score = \frac{Precision\ \times Recall}{Precision + Recall} \tag{4}$$

Sensitivity's formula is stated as in (5).

$$Sensitivity = \frac{True\ Positives}{True\ Positives + False\ Negatives} \tag{5}$$

Specificity's formula is stated as (6).

$$Specificity = \frac{True\ Negatives}{True\ Negatives + False\ Positives} \tag{6}$$

3 Results and Discussion

The findings and discussion are presented in this section. Experiments were conducted using methodologies that included comparing five models to identify the best one based on performance and assessing the impact of feature representation on the dataset. The results from each model are presented in Table 8.

Table 8. A comparison of CNN-based architectures performance evaluated on testing dataset

Models	Loss	Accuracy %	Recall %	Precision %	F1 score %	Sensitivity %	Specificity %
Custom CNN	0.4375	91.79	91.79	91.79	82.08	96.69	98.40
VGG-16	8.3716	76.40	76.28	76.37	38.89	78.34	89.45
EfficientNetB0	0.8382	82.90	82.78	82.88	62.60	94.41	97.55
ResNet50	0.9286	86.20	85.97	86.17	63.84	92.82	96.69
LeNet-5	0.6229	76.28	76.28	76.28	28.85	76.28	88.14

From Table 8, it can be seen that the Custom CNN model demonstrated superior performance in the classification of cocoa diseases. Exhibiting the lowest loss value at 0.4375, the model's predictions closely aligned with the actual labels. It achieved the highest accuracy among the evaluated models at 91.79%, reflecting its robustness in correctly identifying both healthy and diseased cocoa plants. The model's recall and sensitivity were equally impressive, registering at 91.79% and 96.69%, respectively, underscoring its effectiveness in detecting true positive cases of disease. Precision matched the recall at 91.79%, suggesting a high likelihood of accurate disease predictions. While the F1 score stood at a substantial 82.08%, it was slightly lower than other metrics, indicating a nuanced balance between precision and recall. Notably, the model's specificity was the highest at 98.40%, pointing to its exceptional capability in accurately identifying healthy plants and minimizing false positives.

The VGG-16 model's performance was analyzed with less favorable outcomes compared to the Custom CNN. The model's higher loss value at 8.3716 signaled less accurate predictions. With a moderate accuracy of 76.40%, VGG-16 demonstrated a greater propensity for misclassification of both healthy and diseased cocoa plants. The model's recall and sensitivity were recorded at 76.28% and 78.34%, respectively, which, while reasonable, fell short of the Custom CNN's benchmarks, indicating challenges in the model's capacity to identify all positive cases reliably. A precision of 76.37% further revealed a higher incidence of false positives. The notably lower F1 score of 38.89% highlighted a significant imbalance between precision and recall, pointing to disparities in the model's predictive consistency. Additionally, a specificity of 89.45% suggested reduced efficacy in accurately ruling out healthy plants, marking a distinct area for potential improvement.

The EfficientNetB0 model was also assessed, demonstrating commendable predictive capabilities. The model presented a moderate loss of 0.8382, indicating more accurate predictions than VGG-16 but not quite as precise as the Custom CNN. It achieved a respectable accuracy rate of 82.90%, surpassing VGG-16 and underscoring its reliable classification ability, albeit not at the level of the Custom CNN. EfficientNetB0 showcased a strong recall of 82.78% and an even more remarkable sensitivity at 94.41%, reflecting its proficiency in correctly identifying a high percentage of actual disease instances. With a precision of 82.88%, the model assured a good likelihood of its disease predictions being correct. The F1 score, standing at 62.60%, was significantly higher than that of VGG-16 and indicated a more harmonious balance between precision and recall. The specificity was noted at 97.55%, denoting a high accuracy in correctly classifying healthy plants, though marginally trailing behind the Custom CNN.

The analysis of the ResNet50 model within the framework of this work yielded informative insights. The model registered a loss of 0.9286, which, while exceeding that of the Custom CNN and EfficientNetB0, suggested that its predictions, although less precise, were still generally accurate. ResNet50's accuracy was noted at 86.20%, outperforming VGG-16 and LeNet-5, thereby reliably identifying a substantial majority of both healthy and diseased cocoa plants, albeit not as effectively as the Custom CNN or EfficientNetB0. The

model demonstrated a high recall of 85.97% and sensitivity at 92.82%, signifying its adeptness in correctly recognizing diseased specimens, with sensitivity slightly below EfficientNetB0. Precision was recorded at a commendable 86.17%, indicating fewer false positives in comparison to VGG-16. The F1 score, at 63.84%, was marginally superior to EfficientNetB0, reflecting a well-maintained equilibrium between precision and recall. Lastly, the specificity stood at a robust 96.69%, showcasing the model's capability to accurately identify healthy plants, though not quite reaching the benchmark set by the Custom CNN or EfficientNetB0.

Finally, LeNet-5 exhibited a moderate loss of 0.6229, suggesting relatively accurate predictions, though not as precise as those of the Custom CNN or EfficientNetB0. Its accuracy stood at 76.28%, mirroring that of VGG-16 and indicating less effective classification of both healthy and diseased plants in comparison to the more advanced models. LeNet-5's recall and sensitivity, both at 76.28%, were the lowest recorded among the models, pointing to a deficiency in accurately identifying true positive cases. Similarly, the precision of 76.28% hinted at a greater occurrence of false positives. Notably, the F1 score was the lowest at 28.85%, reflecting a considerable imbalance between precision and recall, underscoring the need for improved model training or architecture adjustments. Lastly, the model's specificity was the least at 88.14%, indicating a comparatively lower success rate in correctly identifying healthy specimens.

A critical insight from the study is the significance of the F1 score, which represents the equilibrium between recall and precision, essential in scenarios with high stakes for false predictions. Notably, despite the dataset's imbalance—with Pod Borer class being significantly outnumbered by Healthy and Black Pod Rot classes—the leading models maintained commendable specificity and sensitivity, a testament to their robustness against skewed class distributions.

4 Conclusion

This study investigated the classification of cocoa diseases using five models: custom CNN, VGG16, EfficientNetB0, ResNet50, and LeNet-5. In summary, the comprehensive analysis presented in this work clearly demonstrates the Custom CNN model's superior performance over its counterparts in diagnosing cocoa plant diseases. Excelling across nearly all evaluated metrics, demonstrated by its performance metrics: it achieved an accuracy, precision, and recall all at 91.79%, followed by an F1 score of 82.08%. Furthermore, the model showed high sensitivity and specificity, at 96.69% and 98.40% respectively, highlighting its robustness in accurately distinguishing between healthy and diseased classes. EfficientNetB0 emerges as a robust alternative, boasting commendable scores that place it just behind the Custom CNN. ResNet50, while noteworthy in its own right, does not quite reach the performance levels of its preceding models. VGG-16 and LeNet-5 round out the comparison as the lesser-performing models, with LeNet-5 registering the lowest scores across the evaluated metrics, indicating a need for significant improvements or potentially a reconsideration of model selection for this specific application. Future studies could explore other pre-trained models on this dataset and implement data augmentation to balance the datasets. Additionally, the impact of different feature representations on performance metrics warrants deeper investigation. Ultimately, this research aims to contribute to a real-time system that reduces losses in cocoa production.

Acknowledgement

This research was funded by the Research Management Centre (PPP), Universiti Malaysia Sabah, under grant number DN20087.

References

1. V. Tiwari, R. C. Joshi, M. K. Dutta, "Dense convolutional neural networks based multiclass plant disease detection and classification using leaf images," Ecol. Inform. 63, 101289 (2021).
2. Z. Wang, C. Tang, X. Sima, L. Zhang, "Research on application of deep learning algorithm in image classification," in Proc. IEEE Asia-Pacific Conf. Image Process. Electron. Comput. (IPEC), 2021, 1122-1125.
3. A. A. Hernandez, J. L. Bombasi, A. C. Lagman, "Classification of sugarcane leaf disease using deep learning algorithms," in Proc. IEEE 13th Control Syst. Grad. Res. Colloq. (ICSGRC), 2022, 47-50.
4. S. Sladojevic, M. Arsenovic, A. Anderla, D. Culibrk, D. Stefanovic, "Deep neural networks based recognition of plant diseases by leaf image classification," Comput. Intell. Neurosci. 2016, (2016).
5. K. He, X. Zhang, S. Ren, J. Sun, "Deep residual learning for image recognition," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit., 2016, 770-778.
6. A. V. Panchal, S. C. Patel, K. Bagyalakshmi, P. Kumar, I. R. Khan, M. Soni, "Image-based plant diseases detection using deep learning," Mater. Today Proc. 80, 3500-3506 (2023).
7. S. V. Militante, B. D. Gerardo, N. V. Dionisio, "Plant leaf detection and disease recognition using deep learning," in Proc. IEEE Eurasia Conf. IOT, Commun. Eng. (ECICE), 2019, 579-582.
8. K. Schindler, "Cocoa segmentation in satellite images with deep learning," 2018.
9. R. A. Godmalin, C. J. Aliac, L. Feliscuzo, "Classification of cacao pod if healthy or attack by pest or black pod disease using deep learning algorithm," in Proc. IEEE Int. Conf. Artif. Intell. Eng. Technol. (IICAIET), 2022, 1-5.
10. K. Akshai, J. Anitha, "Plant disease classification using deep learning," in Proc. 3rd Int. Conf. Signal Process. Commun. (ICPSC), 2021, 407-411.
11. M. Agarwal, S. K. Gupta, K. Biswas, "Development of efficient cnn model for tomato crop disease identification," Sustain. Comput. Inform. Syst. 28, 100407 (2020).
12. M. Yildirim, A. Cinar, E. Cengil, "Classification of flower species using cnn models, subspace discriminant, and nca," in Proc. Int. Conf. Innov. Intell. Inform. Comput. Technol. (3ICT), 2021, 334-339.
13. B. Samia, Z. Soraya, M. Malika, "Fashion images classification using machine learning, deep learning and transfer learning models," in Proc. 7th Int. Conf. Image Signal Process. Appl. (ISPA), 2022, 1-5.
14. R. Shakil, B. Akter, F. Faisal, T. R. Chowdhury, T. Roy, A. Khater, "A promising prediction of diabetes using a deep learning approach," in Proc. 6th Int. Conf. Comput. Methodol. Commun. (ICCMC), 2022, 923-927.
15. S. Vani, T. M. Rao, "An experimental approach towards the performance assessment of various optimizers on convolutional neural network," in Proc. 3rd Int. Conf. Trends Electron. Inform. (ICOEI), 2019, 331-336.

16. A. Ganguly, R. Das, S. Setua, "Histopathological image and lymphoma image classification using customized deep learning models and different optimization algorithms," in Proc. 11th Int. Conf. Comput. Commun. Netw. Technol. (ICCCNT), 2020, 1-7.
17. Y. Tang, Y. Wang, J. Li, W. Zhang, L. Wang, X. Zhai, A. Han, "Classification of Chinese Herbal Medicines by deep neural network based on orthogonal design," in Proc. IEEE 4th Adv. Inform. Manag. Commun. Electron. Autom. Control Conf. (IMCEC), 2021, Vol. 4, 574-583.