INTERNATIONAL JOURNAL OF ADVANCE RESEARCH, IDEAS AND INNOVATIONS IN TECHNOLOGY

ISSN: 2454-132X
Impact Factor: 6.078
(Volume 10, Issue 6 - V10I6-1395)
Available online at: https://www.ijariit.com

A Hyperparameter Tuning Optimized Convolutional Neural Network for Classification of Fruit Type and Quality through Computer Vision

Aarav Kodathala Reddy
kaaravreddy005@gmail.com
Independent Researcher, Mumbai, Maharashtra

ABSTRACT

Food poisoning is a problem affecting people on a global scale, killing 420,000 people a year as of 2022. This problem is exacerbated by the distribution of already-rotten food from farms to vendors and can be mitigated by preventing infected food from ever leaving farms in the first place, or by identifying rotten foods before vendors sell them. Thus, this study summarizes the building of a hyperparameter-optimized deep learning model that uses a Convolutional Neural Network (CNN) to identify the kind of fruit and its quality by looking at an image of a fruit, a simple process. This automation of food classification and safety allows lower-income farmers and vendors to escape the time and monetary cost of manually verifying whether or not each fruit they distribute/sell is safe to eat.

Keywords: *Food safety, Spoilage, Hyperparameter-optimized deep learning model, Convolutional neural network, Computer vision*

1. INTRODUCTION

Food safety and nutrition are closely interconnected. Approximately, 600 million people globally fall ill from contaminated food, leading to 420,000 deaths. Unsafe food causes a loss of US$ 110 billion annually in low- and middle-income countries [1]. This disproportionately affects families with young children, senior citizens, and pregnant/chronically ill individuals, making it an essential and urgent issue to address [2]. However, most farmers and local vendors are unable to access adequate tools to perform the task in a timely, accurate, and affordable manner, leading to perpetuation of spoiled foods among consumers [3]. In some specific scenarios, particular chemical reagents with specific purposes-like fluorescent probes are also employed for fruit freshness testing [4]. While these technologies have proven to be efficient in detecting fruit freshness, they all have some disadvantages in common, such as high investment costs, limited capacity to process large-scale data, and reliance on human experience.

With the evolution of artificial intelligence, traditional methods have been progressively replaced by machine learning approaches in detection of fruit freshness. These approaches have been garnered for their non-invasive nature, rapidity, precision, and low-cost benefits [5]. For instance, machine learning techniques like Support Vector Machines (SVM) and decision trees have been utilized in spectroscopic profile datasets of fruits to determine their freshness [6]. Classic deep learning models, such as GoogLeNet, have been integrated with dimensionality reduction techniques to assess the fruit image data for freshness detection [7]. In case of increasingly complicated applications, larger deep learning models, such as YOLO, have been used for freshness detection [8]. In a study, a Convolutional Neural Network (CNN) multi-task learning (MTL) model has been utilized for computer vision-based fruit freshness detection [9].

Computer vision has the potential to address these concerns and has already been explored in several research initiatives in the past. However, these academic attempts often face limitations such as insufficient data (only 5,658 images) [10], which leads to overfitting on a small dataset, or a limited variety of fruits (only three types of fruits) [11], restricting their applicability. Research on applying CNN to fruit freshness detection remains limited and needs further investigation. This study aims to explore the potential of using CNN, optimized using hyperparameter tuning approach, for this purpose. The CNN algorithm was chosen specifically because of its specialized nature and high accuracy when it comes to image classification with large datasets [3]. This model's pipeline also conclusively tested how different combinations of hyperparameter-pairs work in tandem, ultimately culminating in the selection of features that function well not just in hyper-specific cases, but also when combined with other well-functioning features that impact a different part of the algorithm.

2. MATERIALS AND METHODS

i. Dataset acquisition

This research was conducted on a MacBook Pro using the Apple M3 Pro chip, an 11-core central processing unit and a 14-core graphics processing unit (GPU), using 18 gigabytes of random-access memory. The datasets were obtained from various sources including published articles [12], [13], Kaggle [14], [15], [16], [17], and open source libraries with computer vision datasets [18]. The details of the dataset are presented in Table-1. The dataset contained images of five fruits, distinguished into fresh and rotten fruits, accounting for a total of ten classes as shown in Table-1 and Chart-1. The following sections briefly outline how the datasets were segregated, converted, and inputted into the model for training and validation using the various libraries. The detailed process involved in this research is illustrated in Chart-2.

Table-1: Summary of dataset.

Class	Number of Images
Healthy Apple	2,782
Rotten Apple	2,650
Healthy Banana	2,927
Rotten Banana	2,889
Healthy Mango	2,794
Rotten Mango	2,856
Healthy Orange	2,952
Rotten Orange	2,927
Healthy Strawberry	2,865
Rotten Strawberry	2,867
Total Images	28,509

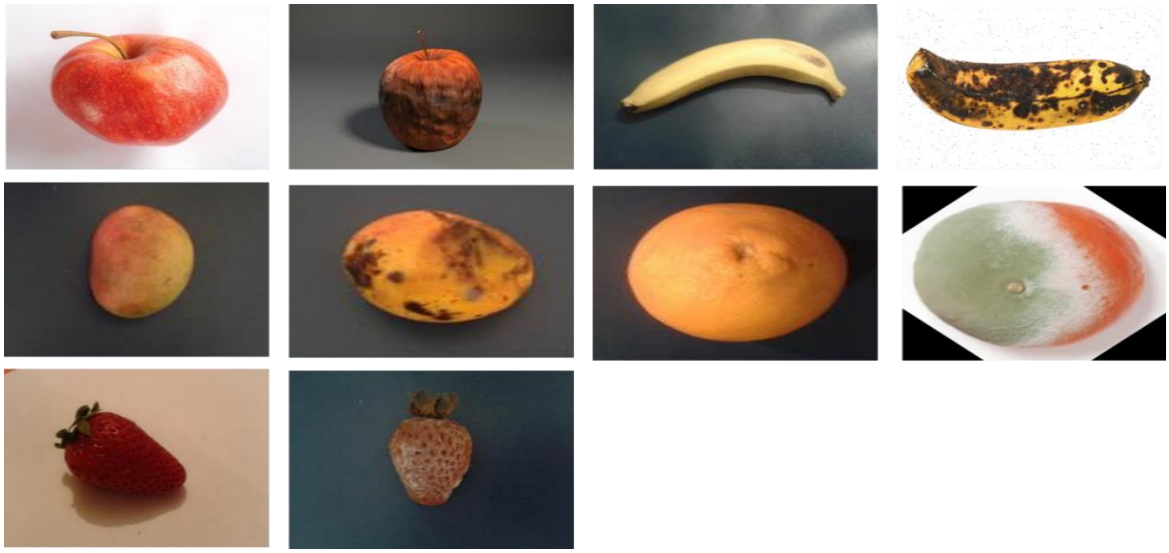

Chart-1: Dataset sample

ii. External libraries

External libraries used in this model were NumPy 1.26.4, iPython 8.25.0 and SciKit Learn 1.4.2. NumPy was used to work with arrays, iPython was used in building the program, and SciKit Learn inclusive of several machine learning models was used for integrating with other modules to accomplish its task. Tensorflow 2.17.0 (with Keras 3.4.1) was used as an external library in the proposed model to help with pre-processing the data, building the proposed model, and for training the model to formulate predictions.

iii. Data sorting

Data retrieved was sorted into separate classes. Each class contained approximately equal number of images, with difference between the classes with most images and least images limited to only 300 images. In the case of healthy and rotten classes of each fruit, the difference between the healthy and rotten class was limited to a maximum of 130 images, aimed at preventing overfitting of data in any of the classes. Data consisted of images gathered from varying camera lens angles and qualities. This was done to ensure applicability; by taking into consideration the challenges individuals may face with accessing good quality camera.

iv. Data conversion to arrays

Before splitting the data, the list of images and their classes were converted into NumPy arrays using the numpy.array() function. This converted the list to an n-dimensional array that served the function of a list while being up to 50 times faster to process than a traditional Python list [19].

Each variable was iterated through using a "for" loop which, for each image in the subdirectory, appends the image itself (resized to a size of 32x32 and converted to a NumPy array) to an empty list using tensorflow.keras.utils.load_img function, which loaded the image into a Python Imaging Library (PIL) or a PIL format for Python to read.

Then, this appended the image's class to another empty list [20]. However, to make the data possible for the model to fit to, the images' classes names were replaced by numbers from 0-9, in order of their position as presented in Table-1, from first to last.

v. Data segmentation

Data were split-up into images and respective labels using Python glob module (short for "global"). The Python glob module found all the pathnames that matched with a specified pattern. In the proposed model, this assigned the data in each class's subdirectory to a variable through glob.glob() function, which returned all the matched pathnames [21]. NumPy used a novel way to manipulate data through sklearn.model_selection.train_test_split to split each image in the dataset and its respective class for data training and validation using the parameters of the image data and the classes. Additionally, two parameters including test_size and random_state were used. The test_size provided a ratio for the distribution of fruit images for "training" and "validation" purposes (3:1 ratio). While the random_state of the data provided a fixed degree of shuffling, ensuring that the data was randomized to prevent the model from overfitting and training on only limited classes (e.g., only training on groups of rotten apples and oranges before meeting a healthy banana during the validation process), while also ensuring standardization of the shuffling across repeated running of the model over several days [22].

vi. Data transformation

One-hot encoding method was followed to transform the categorical data into numerical format (0 and 1). The y-values for both training and validation were one-hot encoded using keras.utils.to_categorical() function, which transformed the y-data from being a single column with 10 possible values per row, to having 10 different values with two possibles values per cell (0 and 1), corresponding to whether that column was "yes" or "no". Thus, instead of healthy_bananas being represented by a single "2" in the y-data column, they were represented by a "1" on the healthy_bananas column (i.e., instead of "2" in the column), and a "0" in other places [23].

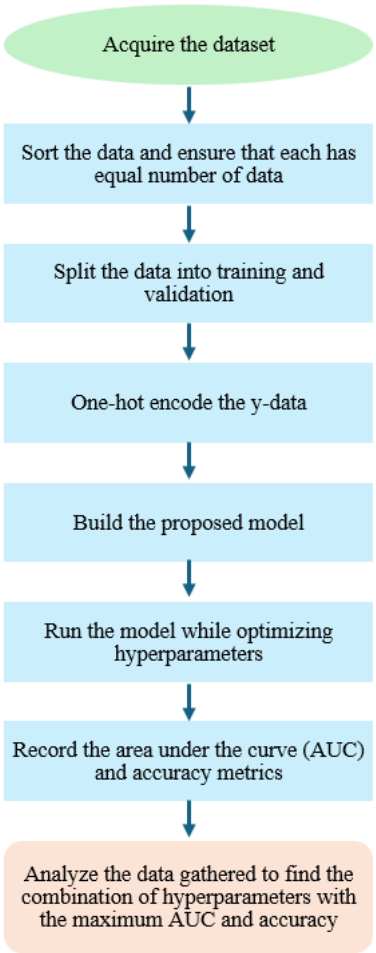

Chart-2: Schematic representation of the process involved in this research

vii. Proposed CNN Model

The actual model was developed once the data was one-hot encoded. In this model, all the hyperparameters were altered during the course of this study. We employed a CNN model, with basic structure comprising a Sequential() model, an input Conv2D layer, two MaxPooling2D layers, a Flatten layer, and two Dense Layers inclusive of a softmax activation function. We used the "he_uniform" kernel initializer through all layers. The metrics (i.e., accuracy and area under the curve [AUC]) remained as constant variables. In this model, the activation functions applied a mathematical function to the output of each neuron, which made the model nonlinear. The activation function computed the weighted sum and added bias, to determine if the neuron should be activated or not.

The weights were fundamentally the strengths of the connections between individual neurons, and biases were constants that were appended to each neuron's output [24].

viii. CNN Architecture

CNN consists of multiple layers, and the architectural design and its parameters are shown in Chart-3 and Chart-4. These include:

a. **Conv2D layer (2D Convolutional layer):** This basically "slides" across the input data of a given image, from left to right, and from top to bottom, capturing all squares of the equivalent size specified within the image. Then, multiplying and condensing them into a single square on the output image [25].

b. **Normalization:** The normalization functions each possess a distinctive way to transform data to be on a similar and fixed scale, with identical range to enhance the model's performance and training strength. This has been used mainly when the X-data gathered was largely variable, as in the case of images [26].

c. **MaxPooling2D:** This is a down sampling technique used in reducing the dimensions of the input it gets in a CNN, driven towards elimination of the amount of stored information in the image, while preserving the important attributes essential for classifying it. This has been helpful in reducing the number of parameters the image has, and in reduction of both the computational power needed to run the model and overfitting data [27].

d. **Flatten() layer:** This layer can transform the multidimensional image data into a single-dimensional array. This has been helpful in passing the data across the model from the preceding convolutional layers that yields multidimensional outputs to its subsequent dense layers that requires one-dimensional inputs [28].

e. **Dense layers:** In these layers, neurons in the previous layer send inputs to each neuron in this layer. This layer assists in the output classification of the fruit image according to the input from convolutional layers, which were resized through flatten() [29].

f. **Loss functions:** Loss functions assist in comparing the target and predicted output values. This helps to find the difference or "loss" between them, basically by yielding the amount the model missed for the actual answer. This functions as a measure of how efficiently the neural network can predict the classes precisely [30].

g. **Optimizer:** This is an algorithm that transforms the model's parameters to decrease the loss function' value, generally by predicting the combination of weights and biases of the neural network that will result in the maximum accuracy [31].

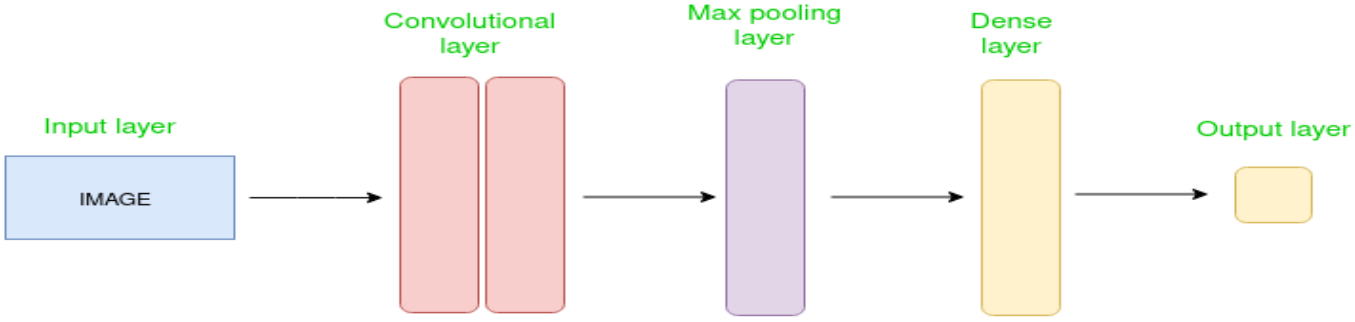

Chart-3: A CNN, simplified [32]

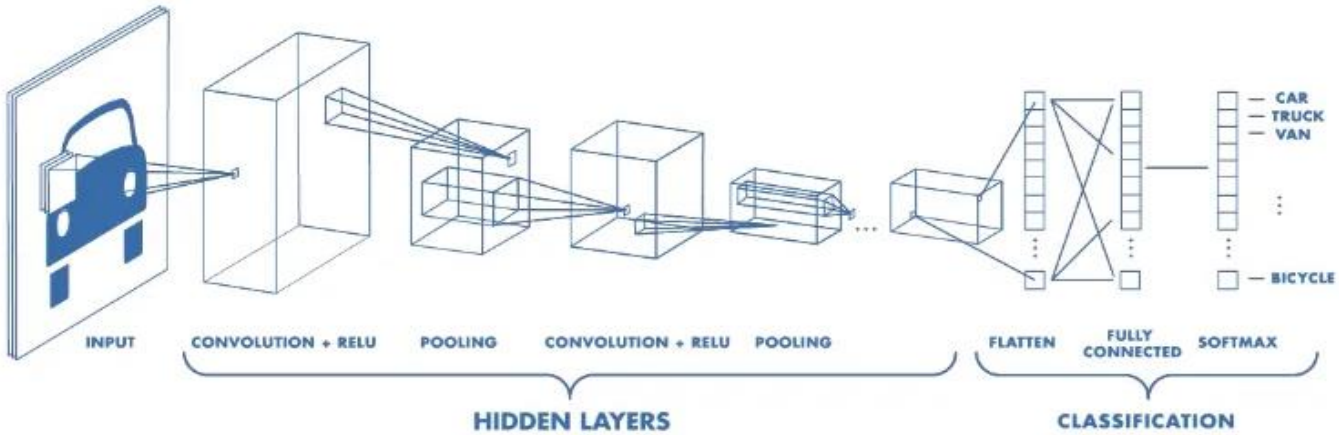

Chart-4: A CNN classifying vehicle images, using ReLu as the activation function for the convolutional layers, and softmax as the activation function for the dense layer [33]

ix. Model's parameters

The parameters such as the x_train data, the y_train data, the x_validation data, the y_validation data, and the batch size remained fixed while the model was run using model.fit. The purpose of the training data was to allow the model to iterate through it multiple times and accordingly update its weights and biases to increase accuracy. The purpose of the validation data was to verify whether the model continued to be accurate on concealed data (as the model was not allowed to update its weights and biases pertaining to validation data) and observe if the model was overfitting, and if the batch size was the number of training samples the model has to work through before its weights and biases were updated. Finally, the number of epochs was the number of times the model should iterate through the entire training dataset, and this was one of the independent variables in this research [34].

x. Performance Metrics

Metrics were used to assess the performance of the model. These include accuracy and area under the Receiver Operating Characteristics (ROC) curve [35]. The metrics are the same as the loss functions, except that they were not considered by the model itself during data training but were observed for comparison of different models, as followed in this study.

Accuracy metric denotes the quotient of the number of correct predictions divided by the total number of predictions made and was usually presented as a decimal. The ROC curve showed the performance of a classification model at all classification thresholds, namely plotting the model's true positive rate (TPR, number of correct "yes" predictions) on the y-axis and false positive rate (FPR, number of incorrect "yes" predictions) on the x-axis, with both axes having a minimum value of 0 and a maximum of 1.

The definition for TPR, a synonym of recall is as follows [36]:

$$TPR = \frac{TP}{TP + FN} \quad \textbf{(1)}$$

The definition for FPR is as follows [36]:

$$FPR = \frac{FP}{FP + TN} \quad \textbf{(2)}$$

The area under the ROC curve used integration to find the AUC value, with a minimum AUC of 0.0 and a maximum AUC of 1.0, the latter of which signifies the most ideal capture rate for the model [36].

xi. Ablation Study

Round 1 Testing

For the first round of testing, the model was run with every possible normalization-epoch number pair that could be formed out of three different types of normalization and three different numbers of epochs. The three epoch lengths that were tested include: 15 epochs, 30 epochs and 45 epochs.

The three normalizations tested were as follows:

a. Batch Normalization:

As the distribution of values along the activation functions constantly changes during the data's training, the process was slowed as each layer needed to adapt to a new distribution. This problem was known as internal covariate shift. Batch normalization solved this problem by forcing the input of every layer to have approximately the same distribution in every training step, using the following processes:

Calculating the mean and variance of the layers input using the below formulas [37]:

$$\mu_B = \frac{1}{m}\sum_{i=1}^{m} x_i \text{ Batch mean} \quad \textbf{(3)}$$

$$\sigma_B^2 = \frac{1}{m}\sum_{i=1}^{m}(x_i - \mu_B)^2 \quad \text{Batch variance, here } \sigma \text{ is standard deviation from mean.} \quad \textbf{(4)}$$

Normalizing the layer inputs using the batch statistics calculated in step A as below [37]:

$$\bar{x_i} = \frac{x_i - \mu_B}{\sqrt{\sigma_B^2 + \epsilon}} \quad \textbf{(5)}$$

Scaling and shifting to obtain the output of the layer by using the below formula [37]. Please note that γ and β were learned during training along with the original parameters of the network.

$$y_i = \gamma \bar{x_\iota} + \beta \quad \textbf{(6)}$$

b. Layer Normalization:

This works across every batch in a layer and ensures that all neurons in a particular layer effectively have the same distribution across all features for a given input. This removes the dependence on batches that batch normalization has, allowing the use of smaller batches or batches of varying size, unlike in batch normalization. Note that if each image of a fruit has d features, it's a d-dimensional vector.

If there were B elements in a batch, layer normalization was done along the length of the d-dimensional vector but not across the batch of size B. The following are the steps involved in layer normalization [38].

$$\mu_l = \frac{1}{d}\sum_{i=1}^{d} x_i \quad \textbf{(7)}$$

$$\sigma_l^2 = \frac{1}{d}\sum_{i=1}^{d}(x_i - \mu_l)^2 \quad \textbf{(8)}$$

$$\hat{x}_i = \frac{x_i - \mu_l}{\sqrt{\sigma_l^2}} \quad \textbf{(9)}$$

$$or\ \hat{x}_i = \frac{x_i - \mu_l}{\sigma_l^2 + \epsilon}$$

Adding ϵ helps when σ_l^2 is small **(9)**

$$y_i = LN(x_i) = \gamma \cdot \hat{x}_i + \beta \quad \textbf{(10)}$$

These steps remained near-identical to those undertaken during batch normalization. However, instead of the batch's statistics, we used the mean and variance corresponding to the specific input to the neurons in a particular layer "k". This was equivalent to normalizing the output vector from the layer "k-1" [38].

c. Group Normalization:

This form of normalization divided the channels into groups and normalized the features within each group (channels were a specific map of attributes of the image produced by applying filters to the input data through convolution). Group normalization does not exploit the batch dimension, and its computation was independent of batch sizes. It also conducts the normalization, for example: unlike batch normalization which normalizes across examples in a batch. The main idea behind group normalization was to provide a normalization technique that remains independent of the batch size. This can be beneficial in cases where each data sample has high memory-consumption. The following are the steps involved in group normalization [39].

$$\mu_i = \frac{1}{m}\sum_{k \in S_i} x_k \quad \textbf{(11)}$$

$$\sigma_i^2 = \frac{1}{m}\sum_{k \in S_i}(x_k - \mu_i)^2 \quad \textbf{(12)}$$

$$\hat{x}_i = \frac{x_i - \mu_i}{\sqrt{\sigma_i^2 + \epsilon}} \quad \textbf{(13)}$$

In the above equations, i is an index and x is the feature computed by the layer. A group norm layer formally computes μ and σ in a set S_i defined as: $S_i = \left\{k \mid k_N = i_N, \left\lfloor \frac{k_c}{C/G} \right\rfloor = \left\lfloor \frac{I_c}{C/G} \right\rfloor\right\}$. In this definition, G is the number of groups, which is a predefined hyperparameter (G=32) by default. The number of channels per groups is represented by C/G, and $\lfloor$ is the floor operation. The last term here means that the indexes i and k are in the same group of channels, believing each group of channels are stored in a sequence along the C axis [39].

Round 2 Testing

For the second round of testing, the model was run with every possible activation function-optimizer pair that could be formed out of three different activation functions and three different optimizers.

The three activation functions that were tested are as follows:

a. Rectified Linear Unit (ReLu):

This was a piece-wise linear function that will output the input directly if it was positive; otherwise, it will output zero. This activation function was used to help the model train easily and in achieving better performance. It involved simple computations, with an output value equal to zero and worked like a linear function, making it easier to optimize. The formula employed was:

f(x) = max(0,x) with a range of 0 to infinity [40]. **(14)**

b. Hyperbolic Tangent Function (Tanh):

The tanh function yielded values in the range of -1 to +1 and was zero-centered. This showed that the output of tanh functions was symmetric over the origin of the coordinate system, which has been considered an advantage as it could aid the learning algorithm converge faster. Nonetheless, the tanh has some limitations including its gradients that can become very small (i.e., closer to zero) during backpropagation. This limitation can be challenging, especially for deep networks with several layers as the gradients of the loss function may become too small to produce substantial changes in the weights during training as they backpropagate to the initial layers. This problem has been known as the vanishing gradient problem, which can largely reduce the training process, leading to poor convergence.

The function is defined by:

f(x) = (e^x - e^-x) / (e^x + e^-x), with a range between -1 and +1 [41]. **(15)**

c. Sigmoid Function:

The sigmoid function was a smooth, continuously differentiable function, which carried a real-valued input and squashed it to a value between 0 and 1. It had an "S"-shaped curve that asymptotes to 0 for large negative numbers and 1 for large positive numbers. The function allowed for efficient backpropagation training but suffered from a vanishing gradient problem as well. It can be defined as:

f(x) = 1 / (1 + e^-x), with a gradient between 0 and 1 [41]. **(16)**

The three optimizers tested were as follows:

a) Root Mean Square Propagation (RMSProp):

RMSProp was a gradient descent algorithm that worked by calculating the gradient of the loss function with respect to the model's parameters and updating the parameters in the opposite direction of the gradient to minimize the loss. RMSProp used an exponential decay that discards history from the extreme past so that it could converge rapidly after finding a convex bowl (a loss function with a single minimum point) [42].

The RMSProp algorithm updated the parameters using the following equations:

Calculating the gradient:

gt = ∇θ J(θ), where J(θ) is the loss function **(17)**

Accumulating squared gradients:

E[g²]t = βE[g²]t-1 + (1-β)gt², where β is the decay rate, typically set to 0.9 **(18)**

Computing the adaptive learning rate:

ηt = η / √(E[g²]t + ε),

where η is the initial learning rate and ε is a small constant to prevent division by zero, often set to 1e-8 **(19)**

Updating the parameters:

θt+1 = θt - ηt * **gt** **(20)**

These steps were repeated for each parameter in the network until convergence or the maximum number of iterations were reached [43].

b) Stochastic Gradient Descent (SGD):

The term "stochastic" in SGD refers to the random selection introducing randomness into the optimization process. In SGD, a single random training example (or a small batch) was selected to calculate the gradient and update the model parameters, instead of the entire dataset as used in traditional methods, whereby the computational cost per iteration can be reduced [44]. The advantage of using SGD is its computational efficiency, especially when using large datasets. The formulas used are as follows [45]:

$$E(w,b) = \frac{1}{n}\sum_{i=1}^{n} L\left(y_i f\left(x_i\right)\right) + \alpha R(w) \quad \textbf{(21)}$$

For $w\forall \epsilon\ 0$,

$$y_i = \rho + \frac{w}{b}\xi - \frac{1}{v_i}\sum_{n=0}^{i} \max\left(0, \rho - (w, x_i)\right), \quad \textbf{(22)}$$

$$\min_{w,p,\xi} \frac{v}{2}\|w\|^2 + bv\xi - \frac{1}{n}\sum_{i=1}^{n} \min\left((w, x_i) + b\right), \quad \textbf{(23)}$$

$$R(w) = \frac{1}{2}\sum_{j=1}^{m} w_j^2 = \|w\|_2^2, \quad \textbf{(24)}$$

$$R(w) = \sum_{j=1}^{m} |w|j\ , \quad \textbf{(25)}$$

$$R(w) = \frac{\rho}{2}\sum_{j=1}^{n} w_j^2 + (1-\rho\xi)\sum_{j=1}^{n} |\ w|_{j.} \quad \textbf{(26)}$$

c) Adaptive Moment Estimation (Adam):

Adam, an adaptive learning algorithm was used to modify each parameter's learning rate depending on its gradient record. This adjustment facilitated the neural network to learn efficiently altogether. Adam employed an approach called momentum, which speeds up the training by accelerating gradients in the right directions by appending a fraction of the previous gradient to the present one. It also used parts of RMSProp, which helped in controlling overshooting (missing the minimum point) by adjusting step size [46]. The formulas used are as follows [47]:

$$m_t = \beta_1 m_{t-1} + (1-\beta_1)\left[\frac{\delta L}{\delta w_t}\right] \quad \textbf{(27)}$$

$$v_t = \beta_2 v_{t-1} + (1-\beta_2)\left[\frac{\delta L}{\delta w_t}\right]^2 \quad \textbf{(28)}$$

$$\hat{m}_t = \frac{m_t}{1-\beta_1^t}\,\hat{v}_t = \frac{v_t}{1-\beta_2^t} \quad \textbf{(29)}$$

$$w_{t+1} = w_t - \hat{m}_t\left(\frac{\alpha}{\sqrt{\hat{v}_t}+\varepsilon}\right) \quad \textbf{(30)}$$

Round 3 Testing

For the third round of testing, the model was run with every possible Conv2D layer number-loss function pair that could be formed out of three different numbers of Conv2D layers and three different loss functions. The three numbers of Conv2D layers that were tested include: 3 layers, 4 layers and 5 layers.

The three loss functions tested were as follows:

a) Categorical Cross-Entropy:

Categorical cross-entropy is a loss function that was used in multiclass classification tasks, primarily popular because of its probabilistic nature, wherein it allows the model to output the probabilities of each class. However, it required one-hot encoding to function, which acts as a disadvantage. The formula used is as follows [48]:

$$H(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) H(y, \hat{y}) = -\sum_i y_i \log(\hat{y}_i) H(y, \hat{y}_i) = -\sum_i y_i \log(\hat{y}_i) \quad \textbf{(31)}$$

Here y is true probability distribution (usually in the form of one-hot encoded vectors) and $\hat{y}$ is predicted probability distribution [48].

b) Kullback-Leibler divergence (KL Divergence):

KL Divergence is a nonsymmetric metric that measures the relative entropy or difference in information represented by two distributions. It can be assumed as measuring the distance between two data distributions showing how different the two distributions are from each other, and how it is utilized to ensure that input or output data in production doesn't drastically change from a baseline. The baseline can be a training production window of data or a training or validation dataset. The formula used is as follows [49]:

$$D_{KL}(p(x)||q(x)) = \sum_{x \in X} p(x) \ln \frac{p(x)}{q(x)} \quad \textbf{(32)}$$

c) Focal Loss:

This function is a variation of cross-entropy, addressing class imbalance during training. Focal Loss applied a modulating term to the cross-entropy loss in order to focus learning on "hard misclassified" examples. Essentially, this scaling factor automatically down-weighs the contribution of easy examples during training and rapidly focuses the model on hard examples. It is a dynamically scaled cross-entropy loss, where the scaling factor decays to zero as confidence in the correct class increases. Focal Loss is defined as [50]:

$$FL(pt) = -(1-p_t)^{\gamma} \log(p_t) \quad \textbf{(33)}$$

3. RESULTS

i. Data from Round 1 Testing

During round 1, for all tests, the activation function for the convolutional layers was sigmoid; optimizer was RMSProp; loss function was categorical cross-entropy; and number of Conv2D layers were four. The data gathered from round 1 testing are given in Table-2. Based on the validation data analyzed, 15 epochs had the lowest mean accuracy and mean AUC (Area Under ROC Curve). Conversely, 30 epochs had the highest mean accuracy and second highest mean AUC, with 0.007 lower than the top score. Similarly, 45 epochs had the second highest mean accuracy, with 0.026 lower than the top score; and the highest mean AUC. Hence, 30 epochs was selected as the optimal size. Layer normalization had the highest mean accuracy and mean AUC, and was, therefore, selected as the optimal normalization method.

Table-2: Data gathered from Round 1

	Type of normalization		**Batch Normalization**		**Layer Normalization**		**Group Normalization**		**Mean for training**	**Mean for validation**
Number of epochs			**Train**	**Val**	**Train**	**Val**	**Train**	**Val**		
15 epochs		**Accuracy**	0.953	0.8756	0.9722	0.9106	0.9568	0.8454	0.9607	0.8772
		AUC	0.9984	0.9869	0.9993	0.992	0.9986	0.9823	0.9988	0.9871

			Train	Val	Train	Val	Train	Val		
30 epochs		**Accuracy**	0.9755	0.9186	0.9807	0.9201	0.9758	0.8966	0.9773	0.9118
		AUC	0.9993	0.9913	0.9995	0.9913	0.9993	0.9866	0.9994	0.9897
			Train	**Val**	**Train**	**Val**	**Train**	**Val**		
45 epochs		**Accuracy**	0.9807	0.8482	0.9895	0.922	0.9789	0.8859	0.9830	0.8854
		AUC		0.9765	0.9997	0.9881	0.9993	0.9849	0.9995	0.9832
Mean accuracy			0.9697	0.8808	0.9808	0.9176	0.9705	0.8760		
Mean AUC			0.9989	0.9849	0.9995	0.9905	0.9991	0.9846		

These findings suggest that since an epoch size of 30, and layer normalization result in the most accurate model, the data can now be transformed to the best possible degree and could be iterated through the most efficient number of times (within the study' limitations). Next, the three activation functions and three optimizers could be tested, with a total of nine possible combinations between them.

ii. Data from Round 2 Testing

During round 2, for all tests, normalization was layer normalization; number of epochs was 30; loss function was categorical cross-entropy; and number of Conv2D layers were four.

The data gathered in round 2 testing are given in Table-3. Based on the validation data analyzed, ReLu had the highest accuracy and the second highest AUC, with 0.0039 lower than the top AUC. Tanh had the lowest accuracy and the lowest AUC. Sigmoid had the second highest accuracy, with 0.0245 lower than the top accuracy, and the highest AUC. Notably, sigmoid missed out on the top spot more than ReLu missed out on the top spot. This suggests that ReLu would be the best activation function.

RMSProp had the highest accuracy and the lowest AUC, with 0.0021 lower than the top spot for AUC. SGD had the lowest accuracy and the highest AUC. Adam had the second highest accuracy, with 0.0088 lower than the top spot for accuracy; and the second highest AUC, with 0.0014 lower than the top spot for AUC. Therefore, RMSProp could be implied as the best optimizer available, as it had the highest accuracy and a relatively low difference between its AUC and the highest AUC.

Table-3: Data gathered from Round 2

	Optimizer	**RMSProp**		**Stochastic Gradient Descent**		**Adam**		**Mean for training**	**Mean for validation**
Activation Function		**Train**	**Val**	**Train**	**Val**	**Train**	**Val**		
ReLu	**Accuracy**	0.9928	0.9509	0.9947	0.9348	0.9944	0.943	0.9940	0.9429
	AUC	0.9997	0.9878	0.9998	0.9913	0.9998	0.9896	0.9998	0.9896
		Train	**Val**	**Train**	**Val**	**Train**	**Val**		
Tanh	**Accuracy**	0.9658	0.8838	0.9493	0.876	0.9681	0.8741	0.9611	0.8780
	AUC	0.999	0.9859	0.9983	0.99	0.9992	0.9863	0.9988	0.9874
		Train	**Val**	**Train**	**Val**	**Train**	**Val**		
Sigmoid	**Accuracy**	0.987	0.9362	0.9304	0.8916	0.9534	0.9274	0.9569	0.9184
	AUC	0.9996	0.9939	0.9974	0.9927	0.9983	0.9939	0.9984	0.9935
Mean accuracy		0.9819	0.9236	0.9581	0.9008	0.9720	0.9148		
Mean AUC		0.9994	0.9892	0.9985	0.9913	0.9991	0.9899		

These findings emphasize that since a ReLu activation function and an RMSProp Optimizer result in the most accurate model, the model has now found the best way to adjust the weights and biases of its neurons and chooses the most efficient neurons to activate. Next, the three loss functions and three possible numbers of convolutional layers could be tested, with a total of nine possible combinations between them.

iii. Data from Round 3 Testing

During round 3, for all tests, normalization was layer normalization; number of epochs was 30; activation function for the convolutional layers was ReLu; and the optimizer was RMSProp.

The data gathered in round 3 testing are given in Table-4. Based on the validation data analyzed, the model with 3 layers had the highest accuracy and the highest AUC. The model with 4 layers had the second highest accuracy, and the least AUC. The model with 5 layers had the lowest accuracy and the second highest AUC. Hence, 3 layers was chosen as the optimal number of Conv2D layers. The model with categorical cross-entropy had the lowest accuracy and the lowest AUC. The model with KL Divergence had the second highest accuracy and the second highest AUC. The model with Focal Loss had the highest accuracy and the highest AUC. Hence, Focal Loss was chosen as the optimal loss function.

These findings suggest that since a Focal Loss function and three convolutional layers result in the most accurate model, this enabled to find the best series of optimizations (within the study' limitations), and have, thus obtained the result.

Table-4: Data gathered from Round 3

	Loss function		**Categorical cross-entropy**		**KL Divergence**		**Focal Loss**		**Mean for training**	**Mean for validation**
Number of Conv2D Layers			**Train**	**Val**	**Train**	**Val**	**Train**	**Val**		
3		**Accuracy**	0.9957	0.9459	0.9942	0.9533	0.996	0.9496	0.9953	0.9496
		AUC	0.9996	0.9857	0.9992	0.9876	0.9999	0.9949	0.9996	0.9894
			Train	**Val**	**Train**	**Val**	**Train**	**Val**		
4		**Accuracy**	0.9939	0.9298	0.9928	0.9469	0.9942	0.9524	0.9936	0.9430
		AUC	0.9995	0.9809	0.9995	0.9867	1	0.9946	0.9997	0.9874
			Train	**Val**	**Train**	**Val**	**Train**	**Val**		
5		**Accuracy**	0.9898	0.9401	0.9887	0.9334	0.9839	0.9409	0.9875	0.9381
		AUC	0.9995	0.9869	0.9994	0.9869	0.9998	0.9935	0.9996	0.9891
Mean accuracy			0.9931	0.9386	0.9919	0.9445	0.9914	0.9476		
Mean AUC			0.9995	0.9845	0.9994	0.9871	0.9999	0.9943		

4. DISCUSSION

In recent years, CNNs have demonstrated significant promise in addressing the challenge of identifying fresh and spoiled fruits. However, the currently available machine-learning based solutions to detect fruit freshness are not suitable for large-scale use. In this study we aimed to develop and optimize a CNN model for the classification of five popular fruits (i.e., apples, bananas, mangoes, oranges, and strawberries), across 10 distinct classes, using a dataset of 28,509 images. Performance of the model was assessed through both the AUC of the ROC curve and model accuracy, with a focus on optimizing hyperparameters to achieve the best results in these metrics. Unlike previous studies, this research emphasizes hyperparameter optimization rather than comparing different deep learning models, aiming to enhance the model's classification accuracy and AUC through a three-stage process that modifies key components such as activation functions, optimizers, loss functions, normalization methods, and other such features involved in supporting each prediction. In this study, training accuracy of 99.6%, validation accuracy of 94.96%, training AUC of 0.9999 and a validation AUC of 0.9949 were the final output obtained with the optimized hyperparameters comprising layer normalization, 30 epochs, ReLu as the activation function for the convolutional layers, three convolutional (Conv2D) layers and Focal Loss as the loss of function.

In a study, Miah et al. used a dataset of 5,658 images to discriminate fresh and spoiled fruits across various categories, with the model achieving a validation accuracy of 97.34% [51]. In another study by Nirmaladevi and Kiruthika, the focus was on classifying a narrower range of fruit categories, namely apples, bananas, and tangerines, using a dataset of 9,000 images [10]. This study reported a slightly higher validation accuracy of 98.89%. However, the limited number of fruit classes restricts the applicability of the model to broader real-world scenarios. Additionally, the reliance on accuracy alone as the performance metric may not provide a comprehensive view of the model's reliability. In contrast, our model leverages a much larger dataset of 28,509 images, offering a greater diversity of training and validation data. Furthermore, we assessed our model using both accuracy and the AUC metric, allowing for a more thorough evaluation of its classification performance and ability to distinguish between classes. This multidimensional approach ensures that our model's predictions are not only precise but also generalizable. Additionally, by including five fruit classes- apples, bananas, mangoes, oranges, and strawberries, our model demonstrated superior versatility. The expanded dataset and the inclusion of diverse fruit types make our model more universally applicable for fruit classification and freshness assessment tasks, enhancing its reliability beyond the scope of the previous studies.

The inclusion of the AUC metric as an additional performance measure sets our model apart by providing valuable insights into its ability to handle imbalanced datasets and maintain robustness across varying decision-making thresholds, aspects that were not explored in previous studies.

By using a larger dataset with diverse class representation and employing comprehensive evaluation metrics, our model not only achieves competitive accuracy but also ensures consistent performance under different conditions, making it a more reliable and scalable solution for fruit classification and freshness detection. However, our study does have some limitations, including constraints related to the variety of the dataset and the need for further research to validate findings in real-world scenarios. Despite these limitations, our study makes a substantial contribution to the field of food safety by offering vital insights into the detection and prevention of spoiled fruit adulteration.

This research highlights the potential of combining machine learning and hyperspectral imaging, which together presents a promising way forward for developing robust quality assurance methods in the food industry. By advancing these technologies, we can significantly enhance the accuracy and reliability of food safety protocols, ultimately safeguarding consumer health and fostering trust.

5. CONCLUSION

Our research emphasizes the importance of food safety, with a particular focus on addressing safety concerns with fruit spoilage which is a significant public health concern besides impacting consumer's trust. Our findings highlight the promising role of machine learning and hyperspectral imaging in accurately identifying fruit spoilage, with more than 98% accuracy achieved. This optimized model can be utilized to efficiently segregate rotten fruits from healthy ones, while also categorizing the fruits into groups based on the type of fruit they are by using only an image of the fruit, with different camera lens angles and qualities being considered during the dataset's creation. This facilitates not only low- and middle-income farmers but also fruit-vendors and sales representatives, who require the swift and cost-effective strategies to detect fruits that this model shall provide. Future research may be warranted for analysis of larger datasets comprising a wide variety of fruits, with enhanced modeling techniques and further investigating supplementary analytical methods to improve detection capabilities.

6. ACKNOWLEDGEMENT

The author would like to acknowledge Dr. Sudipta Roy, Associate Professor, Artificial Intelligence & Data Science, Jio Institute, India., for the guidance on the approach to the study and structure of this article, and Molecular Connections Analytics Pvt. Ltd., Bengaluru, for the medical writing and editorial support in preparation of this article.

REFERENCES

[1]"Food safety." Accessed: Nov. 27, 2024. [Online]. Available: https://www.who.int/news-room/fact-sheets/detail/food-safety

[2]S. Jaffee, S. Henson, L. Unnevehr, D. Grace, and E. Cassou, The Safe Food Imperative: Accelerating Progress in Low- and Middle-Income Countries. Washington, DC: World Bank, 2019. doi: 10.1596/978-1-4648-1345-0.

[3]N. Sultana, M. Jahan, and M. S. Uddin, "Fresh and Rotten Fruits Dataset for Machine-Based Evaluation of Fruit Quality," vol. 1, Apr. 2022, doi: 10.17632/bdd69gyhv8.1.

[4]S. Gong et al., "Recent design strategies and applications of organic fluorescent probes for food freshness detection," Food Res Int, vol. 174, no. Pt 1, p. 113641, Dec. 2023, doi: 10.1016/j.foodres.2023.113641.

[5]X. Zhong, M. Zhang, T. Tang, B. Adhikari, and Y. Ma, "Advances in intelligent detection, monitoring, and control for preserving the quality of fresh fruits and vegetables in the supply chain," Food Bioscience, vol. 56, p. 103350, Dec. 2023, doi: 10.1016/j.fbio.2023.103350.

[6]P. Baranowski, W. Mazurek, and J. Pastuszka-Woźniak, "Supervised classification of bruised apples with respect to the time after bruising on the basis of hyperspectral imaging data," Postharvest Biology and Technology, vol. 86, pp. 249–258, Dec. 2013, doi: 10.1016/j.postharvbio.2013.07.005.

[7]Y. Yuan and X. Chen, "Vegetable and fruit freshness detection based on deep features and principal component analysis," Current Research in Food Science, vol. 8, p. 100656, Jan. 2024, doi: 10.1016/j.crfs.2023.100656.

[8]E. Salim and Suharjito, "Hyperparameter optimization of YOLOv4 tiny for palm oil fresh fruit bunches maturity detection using genetics algorithms," Smart Agricultural Technology, vol. 6, p. 100364, Dec. 2023, doi: 10.1016/j.atech.2023.100364.

[9]Y. Zhang et al., "Fruit freshness detection based on multi-task convolutional neural network," Curr Res Food Sci, vol. 8, p. 100733, 2024, doi: 10.1016/j.crfs.2024.100733.

[10]J. Nirmaladevi and V. R. Kiruthika, "Food spoilage alert system by deploying deep learning model," International journal of health sciences, vol. 6, no. S1, Art. no. S1, May 2022, doi: 10.53730/ijhs.v6nS1.6874.

[11]A. Chougule, A. Pawar, R. Kamble, J. Mujawar, and A. Bhide, "Recognizing Fresh and Rotten Fruits Using Deep Learning Techniques," in Data Engineering and Intelligent Computing, V. Bhateja, S. C. Satapathy, C. M. Travieso-González, and V. N. M. Aradhya, Eds., Singapore: Springer, 2021, pp. 205–212. doi: 10.1007/978-981-16-0171-2_20.

[12]N. Sultana, M. Jahan, and M. S. Uddin, "An extensive dataset for successful recognition of fresh and rotten fruits," Data in Brief, vol. 44, p. 108552, Oct. 2022, doi: 10.1016/j.dib.2022.108552.

[13]C. G. Pachon Suescun, J. O. Pinzón Arenas, and R. Jiménez-Moreno, "Spoiled and fresh fruit inspection dataset," vol. 1, Nov. 2020, doi: 10.17632/6ps7gtp2wg.1.

[14]C.Chauhan,"MangoFruitDDS."Accessed:Nov.25,2024.[Online].Available: https://www.kaggle.com/datasets/warcoder/mangofruitdds

[15]S. S. Nayak, "Fresh and Rotten Classification." Accessed: Nov. 25, 2024. [Online]. Available: https://www.kaggle.com/datasets/swoyam2609/fresh-and-stale-classification

[16]Mukhriddin Mukhiddinov, "Fruits and Vegetables dataset." Accessed: Nov. 25, 2024. [Online]. Available: https://www.kaggle.com/datasets/muhriddinmuxiddinov/fruits-and-vegetables-dataset
[17]M. Subhan, "Fruit and Vegetable Disease (Healthy vs Rotten)." Accessed: Nov. 25, 2024. [Online]. Available: https://www.kaggle.com/datasets/muhammad0subhan/fruit-and-vegetable-disease-healthy-vs-rotten
[18]University of Agriculture Faisalabad, "Fresh-vs-Rotten Mango Dataset," Roboflow. Accessed: Nov. 25, 2024. [Online]. Available: https://universe.roboflow.com/university-of-agriculture-faisalabad-fqroe/fresh-vs-rotten-mango
[19]"Introduction to NumPy," W3 Schools. Accessed: Nov. 25, 2024. [Online]. Available: https://www.w3schools.com/python/numpy/numpy_intro.asp
[20]"tf.keras.utils.load_img | TensorFlow v2.16.1," TensorFlow. Accessed: Nov. 25, 2024. [Online]. Available: https://www.tensorflow.org/api_docs/python/tf/keras/utils/load_img
[21]"glob — Unix style pathname pattern expansion," Python documentation. Accessed: Nov. 25, 2024. [Online]. Available: https://docs.python.org/3/library/glob.html
[22]"train_test_split," scikit-learn. Accessed: Nov. 25, 2024. [Online]. Available: https://scikit-learn.org/dev/modules/generated/sklearn.model_selection.train_test_split.html
[23]"Python Keras | keras.utils.to_categorical()," GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/python-keras-keras-utils-to_categorical/
[24]S. Tiwari, "Activation functions in Neural Networks," GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/activation-functions-neural-networks/
[25]A. Thevenot, "Conv2d: Finally Understand What Happens in the Forward Pass," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://towardsdatascience.com/conv2d-to-finally-understand-what-happens-in-the-forward-pass-1bbaafb0b148
[26]"Numerical data: Normalization | Machine Learning," Google for Developers. Accessed: Nov. 25, 2024. [Online]. Available: https://developers.google.com/machine-learning/crash-course/numerical-data/normalization
[27]"Max Pooling," DeepAI. Accessed: Nov. 25, 2024. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/max-pooling
[28]Keras Team, "Keras documentation: Flatten layer," Keras. Accessed: Nov. 25, 2024. [Online]. Available: https://keras.io/api/layers/reshaping_layers/flatten/
[29]Govinda Dumane, "Introduction to Convolutional Neural Network (CNN) using Tensorflow | Towards Data Science.," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://towardsdatascience.com/introduction-to-convolutional-neural-network-cnn-de73f69c5b83
[30] V. Yathish, "Loss Functions and Their Use In Neural Networks," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://towardsdatascience.com/loss-functions-and-their-use-in-neural-networks-a470e703f1e9
[31] "Optimization Rule in Deep Neural Networks," GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/optimization-rule-in-deep-neural-networks/
[32] "Introduction to Convolution Neural Network," GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/introduction-convolution-neural-network/
[33] M. Mishra, "Convolutional Neural Networks, Explained," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://towardsdatascience.com/convolutional-neural-networks-explained-9cc5188c4939
[34] J. Brownlee, "Difference Between a Batch and an Epoch in a Neural Network," MachineLearningMastery.com. Accessed: Nov. 25, 2024. [Online]. Available: https://www.machinelearningmastery.com/difference-between-a-batch-and-an-epoch/
[35] Keras Team, "Keras documentation: Metrics," Kera. Accessed: Nov. 25, 2024. [Online]. Available: https://keras.io/api/metrics/
[36] "Classification: ROC and AUC," Google for Developers. Accessed: Nov. 25, 2024. [Online]. Available: https://developers.google.com/machine-learning/crash-course/classification/roc-and-auc
[37] F. Peccia, "Batch normalization: theory and how to use it with Tensorflow | Towards Data Science," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://towardsdatascience.com/batch-normalization-theory-and-how-to-use-it-with-tensorflow-1892ca0173ad
[38] Bala Priya C, "Build Better Deep Learning Models with Batch and Layer Normalization | Pinecone." Accessed: Dec. 03, 2024. [Online]. Available: https://www.pinecone.io/learn/batch-layer-normalization/
[39] "Group Normalization," Papers with Code. Accessed: Nov. 25, 2024. [Online]. Available: https://paperswithcode.com/method/group-normalization
[40] J. Brownlee, "A Gentle Introduction to the Rectified Linear Unit (ReLU)," MachineLearningMastery.com. Accessed: Nov. 25, 2024. [Online]. Available: https://www.machinelearningmastery.com/rectified-linear-activation-function-for-deep-learning-neural-networks/
[41] Moez Ali, "Introduction to Activation Functions in Neural Networks | DataCamp." Accessed: Nov. 25, 2024. [Online]. Available: https://www.datacamp.com/tutorial/introduction-to-activation-functions-in-neural-networks
[42] "What is RMSProp? Principles & Advantages," Deepchecks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.deepchecks.com/glossary/rmsprop/
[43] "RMSProp," DeepAI. Accessed: Dec. 03, 2024. [Online]. Available: https://deepai.org/machine-learning-glossary-and-terms/rmsprop
[44] R. Roy, "ML | Stochastic Gradient Descent (SGD)," GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/
[45] S. Nagendram et al., "Stochastic gradient descent optimisation for convolutional neural network for medical image segmentation," Open Life Sci, vol. 18, no. 1, p. 20220665, Aug. 2023, doi: 10.1515/biol-2022-0665.

[46] Rahul Agarwal, "Complete Guide to the Adam Optimization Algorithm," Built In. Accessed: Nov. 25, 2024. [Online]. Available: https://builtin.com/machine-learning/adam-optimization
[47] "What is Adam Optimizer?," GeeksforGeeks. Accessed: Nov. 25, 2024. [Online]. Available: https://www.geeksforgeeks.org/adam-optimizer/
[48] M. Sorokin, "Categorical Cross-Entropy: Unraveling its Potentials in Multi-Class Classification," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://medium.com/@vergotten/categorical-cross-entropy-unraveling-its-potentials-in-multi-class-classification-705129594a01
[49] A. Dhinakaran, "Understanding KL Divergence," Medium. Accessed: Nov. 25, 2024. [Online]. Available: https://towardsdatascience.com/understanding-kl-divergence-f3ddc8dff254
[50] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal Loss for Dense Object Detection," 2017 IEEE International Conference on Computer Vision (ICCV), pp. 2999–3007, Oct. 2017, doi: 10.1109/ICCV.2017.324.
[51] M. S. Miah, T. Tasnuva, M. Islam, M. Keya, Md. R. Rahman, and S. A. Hossain, "An Advanced Method of Identification Fresh and Rotten Fruits using Different Convolutional Neural Networks," in 2021 12th International Conference on Computing Communication and Networking Technologies (ICCCNT), Jul. 2021, pp. 1–7. doi: 10.1109/ICCCNT51525.2021.9580117.