University of Calgary

ENEL 525 Fall 2022 - Final Project

CNN Vegetable Image Classification

Nadia Duarte Amador

December 15th, 2022

Introduction

The aim of this project is to build a Convolutional Neural Network (CNN) to recognize and classify vegetables through image classification. Many food industries around the world still use manual labor to sort and label vegetables. This is a task that can be done in a lower amount of time by using deep learning. Sorting and labeling vegetables with the use of deeping learning, can save time and money to industries by automating this process. For the purpose of this project, 15 different classes of vegetables and a total of 21,000 inputs were applied to the neural network.

Methodology

Dataset

The dataset used for this project is called 'Vegetable Image Dataset' (Israk, 2021), and can be found in [1]. The total number of images for this study was 21,000. Images are of size 224x224 pixels and were downloaded in .jpg format. A number of 15,000, 3,000, and 3,000 images were used for the training, testing and validation respectively. There are 15 classes in the dataset and the vegetables in the images are: bean, bitter gourd, bottle gourd, brinjal, broccoli, cabbage, capsicum, carrot, cauliflower, cucumber, papaya, potato, pumpkin, radish, and tomato. The folder downloaded from Kaggle, contains 3 subfolders corresponding to the training, validation and testing images. Each of these contains subfolders with the names for each class of vegetable.

Network Design

For the network design, I have chosen to use 3 convolution layers with the activation function Relu. The Relu derivative computes faster than other activation functions, by only activating certain neurons at the same time, making it an ideal choice for the network design. The first convolutional layer consists of 64 3x3 kernels and is sensitive to the RGB channel. The input shape has a dimension of 224x224x3, where 224 is the width and height of the image and 3 is the depth RGB. The pooling after the first layer downsampled the data into 32 3x3 kernels for the other two convolutional layers to reduce the number of parameters and allow for a more accurate classification. No padding was used in the pooling stage, and a stride of 1 was used for the filter to move 1 pixel at the time. The network architecture is shown in figure 1.

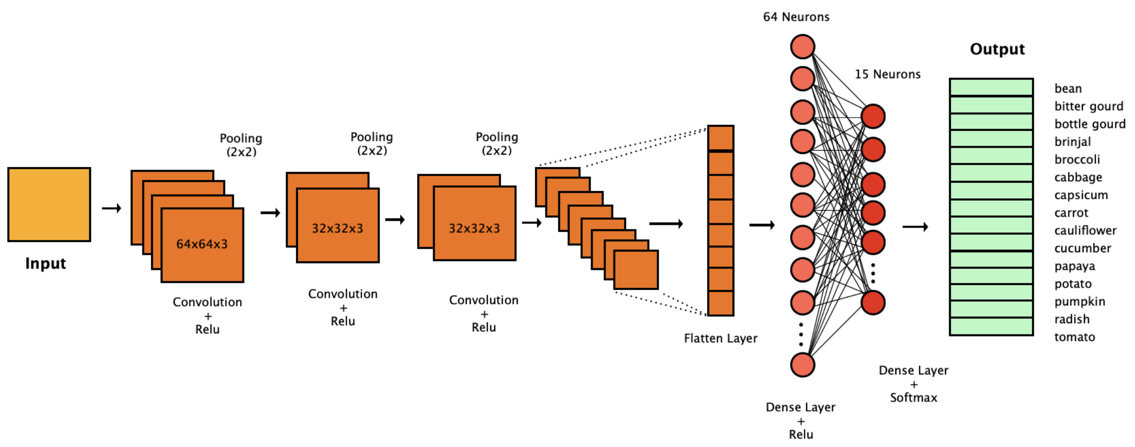


Figure 1, CNN Design

Backpropagation

In CNN, backpropagation allows us to find the loss gradient with respect to the upcoming layer. As seen in figure 2, $\frac{\partial F}{\partial W}$ and $\frac{\partial F}{\partial B}$ will update the parameter values in the filters. The loss gradient for the individual weights can be obtained by using the chain rule as shown in equation 1 and 2.

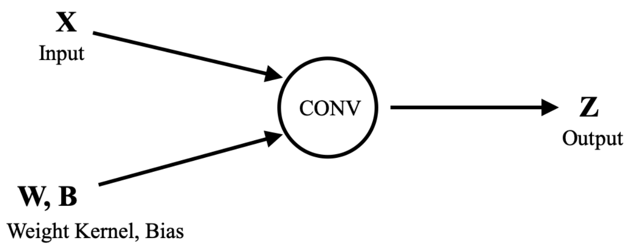


Figure 2, Convolution Diagram

$$\frac{\partial F}{\partial W_{m,n}} = \Sigma_{ij} \frac{\partial F}{\partial Z_{i,j}} \frac{\partial Z_{i,j}}{\partial W_{m,n}} \qquad \textit{(eq. 1)}$$

$$\frac{\partial F}{\partial B_{m,n}} = \Sigma_{ij} \frac{\partial F}{\partial Z_{i,j}} \frac{\partial Z_{i,j}}{\partial B_{m,n}} \qquad \textit{(eq. 2)}$$

After obtaining the gradients by flipping the delta matrix, the cross-correlation operation transforms into convolution. Delta gradient updates are in the form of equation 3, where l is the l^{th} layer and i,j represent the position of a single pixel in the input.

$$\delta^{l}_{i,j} = \frac{\partial F}{\partial W^{l}_{i,j}} \qquad \textit{(eq. 3)}$$

The kernel filter is rotated by 180 degrees by flipping it vertically and horizontally. The convolution operation can be expressed as,

$$\delta^{l+1}_{i',j'} * rot_{180º}\{W^{l+1}_{m,n}\}f'(x^{l}_{i',j'}) \qquad \textit{(eq. 4)}$$

where δ is the sensitivity delta, $W^{l}_{m,n}$ is the weight matrix of layer ɩ, and $f(x^{l}_{i,j})$ is the activation function applied to the convolved input vector at layer ɩ.

By using backpropagation in CNN, we can simplify the network structure and optimize the weights to let the network learn and predict the loss function from the upcoming layer and minimize errors.

Loss Function

For this multi class classification, I have chosen to use the categorical cross entropy loss function. The main function of categorical cross entropy is to take the output probabilities from the softmax function and compare them to the truth values to assign either 0 or 1 to the output by applying one-hot-encoding, so that the index of the target class corresponds to 1 and the other indices correspond to 0. Its purpose is to minimize loss to optimize classification. This function is shown in equation 5, where y is the target vector and y_{test} is the predicted value.

$$L = \sum_{k=0}^{K} - y_{k} log(y_{test,k}) \qquad \textit{(eq. 5)}$$

Training Scheme

A total of 15,000 images out of 21,000 (70% of the data) were used for the training process. For validation purposes, a total of 3,000 images out of 21,000 (15% of the data) were used. For the design of this network, I have chosen to use 10 epochs. Epochs are the number of iterations happening when training the deep learning model. The Adam optimizer in the network is a technique for gradient descent with momentum and RMSP algorithms. The algorithm considers the exponentially weighted average of the gradients to converge faster. Adam provides a high performance and efficient optimization by training the network in a lowest amount of time.

Testing Scheme

A total of 3,000 images out of 21,000 (15% of the data) were used for testing the neural network. All the images used in the network were taken from the Vegetable Image Dataset [1]. No images outside of the dataset were used to test this model. To visualize and verify the data, I have plotted a 5x5 figure showing 25 images taken from the dataset with their corresponding labels. Figure 3, shows the verified data with their corresponding labels.

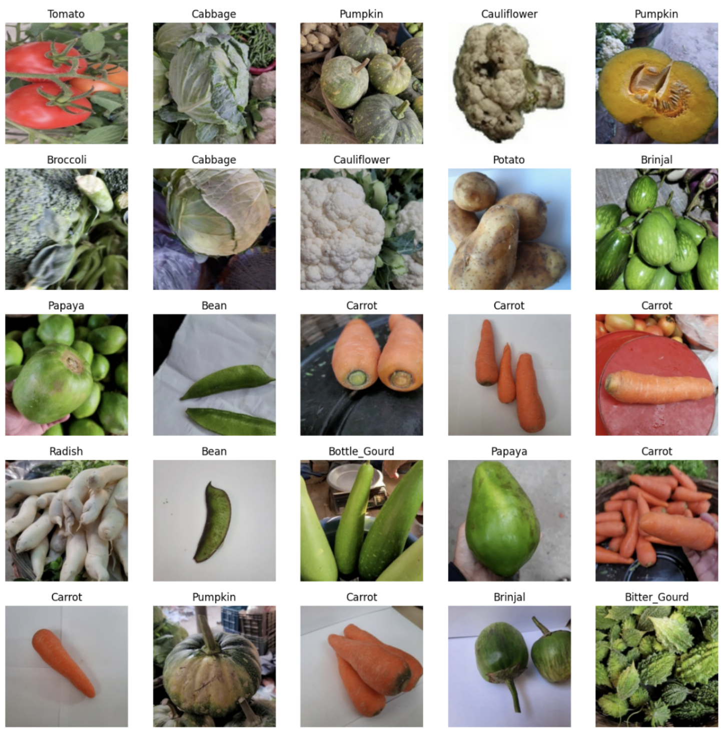


Figure 3, Data Visualization

Results and Discussion

Loss/Validation Curve

The loss was measured per each epoch, with a total of 10 epochs on the x-axis. The training loss curve tells us how well the model fits the current training data, while the validation loss curve tells us how well the model fits new data. The training loss curve descends lower than the validation loss curve with a smooth exponential descent. As we can observe in figure 4, the loss curve is descending for both cases, train and validation. However, it can be observed that the validation loss curve rises after reaching its minimum. This tells us that the model is overfitting, the model fits the training data too well but as the model becomes more complex with adding more data, the generalization error may increase. By reducing the learning rate, the model could be regularized.

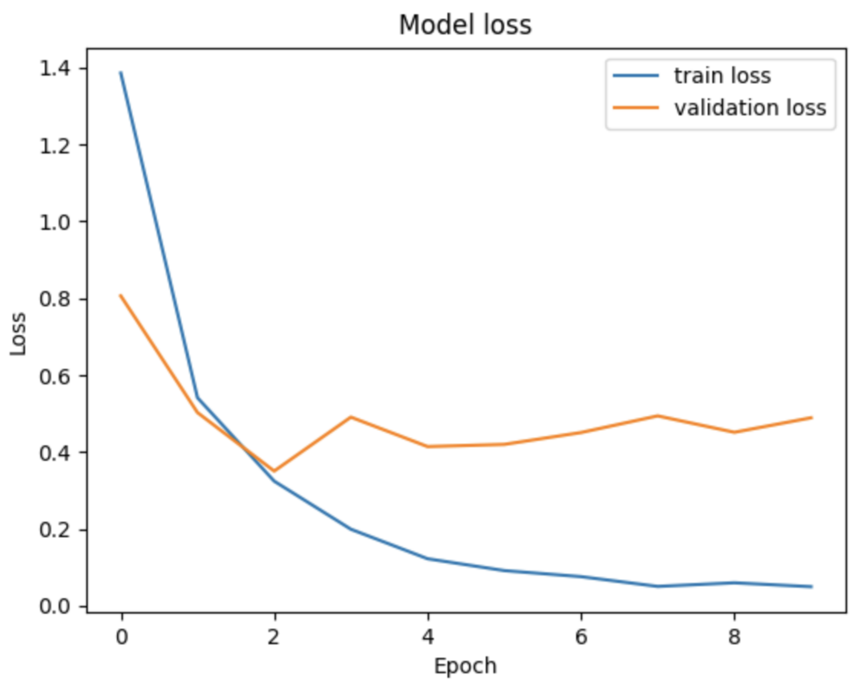


Figure 4, Loss v.s Epoch Model

Table 1, Optimizer, Loss Function and Epochs Summary

Learning Rate	Optimizer	Loss Function	Number of Epochs
0.001 (Default)	Adam	Categorical Cross Entropy	10

Model Summary

For the network design, I have used 3 convolutional layers. The first layer has a dimension of 64x64x3 with an input shape of 224x224x3 from the original image size, the following two layers have a dimension of 32x32x3. Conv2D generates a convolution kernel that produces multi-dimensional arrays (tensor outputs). The function MaxPooling2D downsamples the data to reduce the number of parameters and computations. In max pooling, the maximum of each feature map is calculated to reduce overfitting and the number of computations.

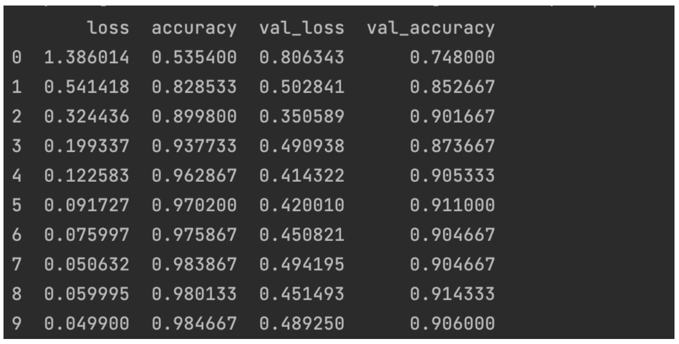
Table 2, Model Summary

Number of Convolutional Layers	Convolution	Pooling	Hidden Layer Activation Functions	Dense Layer Activation Functions
3	Conv2D (3x3)	MaxPooling2D (2x2)	Relu	Relu, Softmax

The hidden layers have a Relu activation function as it requires less computation time and converges faster than other common activation functions. The output dense layer uses the softmax function to predict the multi class probability distribution and classify the output accordingly.

Model Accuracy

The accuracy model in figure 5 shows how in the last epoch an accuracy of 98% was reached. The validation accuracy in the last epoch was 91%. The model performs more efficiently when using the training data with an error difference of 0.07. The results from figure 5 can also be seen in the accuracy plot of figure 6.



```
      loss  accuracy  val_loss  val_accuracy
0  1.386014  0.535400  0.806343      0.748000
1  0.541418  0.828533  0.502841      0.852667
2  0.324436  0.899800  0.350589      0.901667
3  0.199337  0.937733  0.490938      0.873667
4  0.122583  0.962867  0.414322      0.905333
5  0.091727  0.970200  0.420010      0.911000
6  0.075997  0.975867  0.450821      0.904667
7  0.050632  0.983867  0.494195      0.904667
8  0.059995  0.980133  0.451493      0.914333
9  0.049900  0.984667  0.489250      0.906000
```

Figure 5, Model Accuracy Summary

The training accuracy is higher than that of the validation accuracy meaning that the model does not perform as efficiently when new data is entered. The validation accuracy curve does not go over a 0.91 accuracy and the curve is not as smooth as the training accuracy curve.

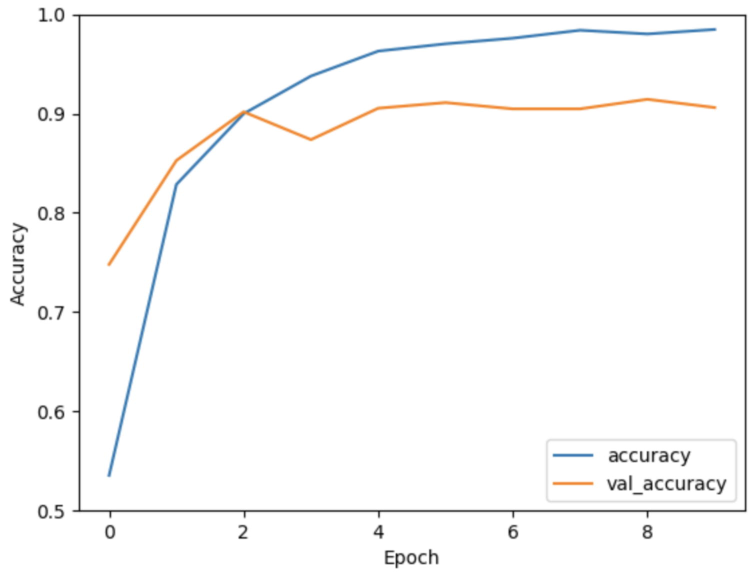
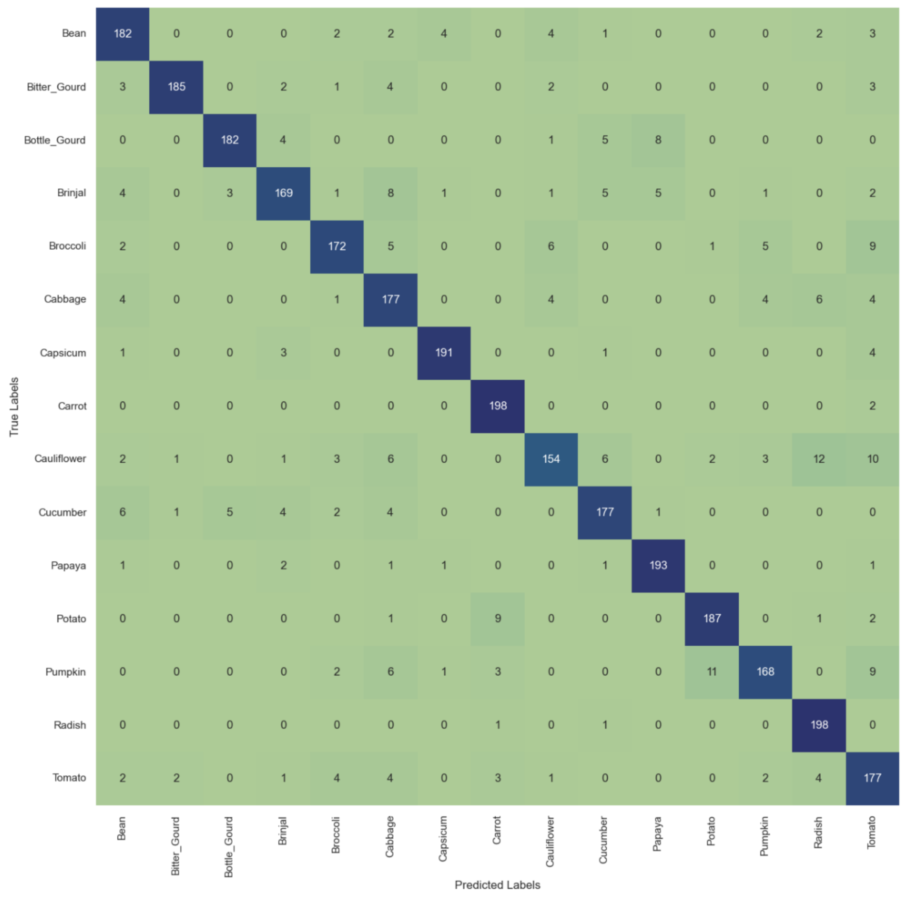


Figure 6, Accuracy Model Curve

Confusion Matrix

The confusion matrix shows how many inputs were correctly and incorrectly classified. In figure 7, the x-axis represents the predicted classes and the y-axis represents the actual classes. Each class had a total of 200 input images corresponding to a specific vegetable from the test set (3,000 images). In (row 1, column 1) from figure 7 we can observe that 182 bean images out of 200 were correctly classified. The other 18 images that were incorrectly classified and are distributed within the broccoli, cabbage, capsicum, cauliflower, cucumber, radish and tomato classes. With 198 images, 99% of the radish and carrot images were correctly classified.

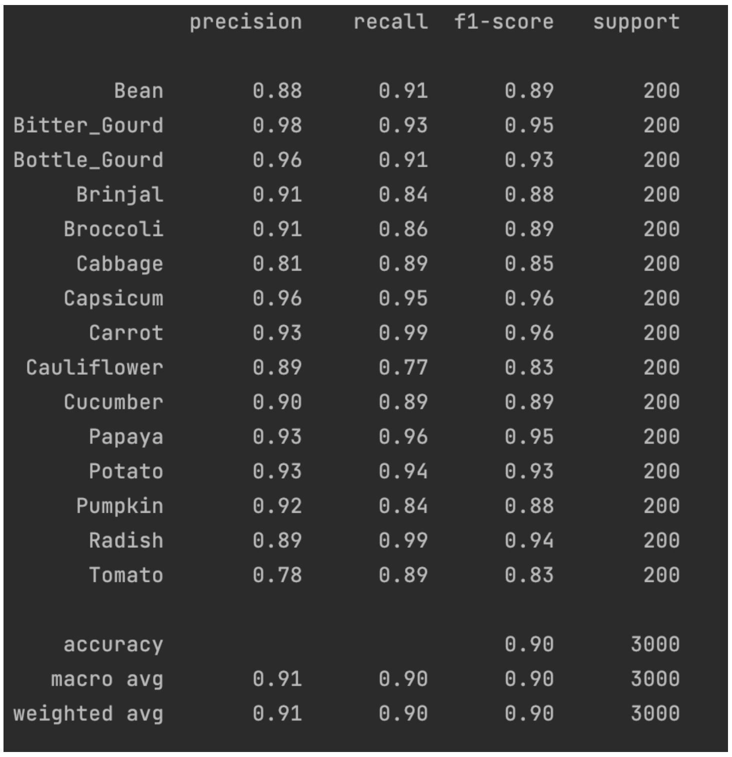


True \ Predicted	Bean	Bitter_Gourd	Bottle_Gourd	Brinjal	Broccoli	Cabbage	Capsicum	Carrot	Cauliflower	Cucumber	Papaya	Potato	Pumpkin	Radish	Tomato
Bean	182	0	0	0	2	2	4	0	4	1	0	0	0	2	3
Bitter_Gourd	3	185	0	2	1	4	0	0	2	0	0	0	0	0	3
Bottle_Gourd	0	0	182	4	0	0	0	0	1	5	8	0	0	0	0
Brinjal	4	0	3	169	1	8	1	0	1	5	5	0	1	0	2
Broccoli	2	0	0	0	172	5	0	0	6	0	0	1	5	0	9
Cabbage	4	0	0	0	1	177	0	0	4	0	0	0	4	6	4
Capsicum	1	0	0	3	0	0	191	0	0	1	0	0	0	0	4
Carrot	0	0	0	0	0	0	0	198	0	0	0	0	0	0	2
Cauliflower	2	1	0	1	3	6	0	0	154	6	0	2	3	12	10
Cucumber	6	1	5	4	2	4	0	0	0	177	1	0	0	0	0
Papaya	1	0	0	2	0	1	1	0	0	1	193	0	0	0	1
Potato	0	0	0	0	0	1	0	9	0	0	0	187	0	1	2
Pumpkin	0	0	0	0	2	6	1	3	0	0	0	11	168	0	9
Radish	0	0	0	0	0	0	0	1	0	1	0	0	0	198	0
Tomato	2	2	0	1	4	4	0	3	1	0	0	0	2	4	177

Figure 7, Confusion Matrix

Network Effectiveness Discussion

Figure 8 shows the precision and recall for the confusion matrix. The network model effectiveness after using the testing data is 90%. However, the accuracy in the training after the last epoch was 98%, which shows a higher performance accuracy than that when using the testing data. As I mentioned before in figure 5, we can predict that the network is going to classify data well if images from outside the dataset were used, however the loss will not be as low and the accuracy will not be as high as that when using the training data. For the loss curve, the training loss curve will continue to decrease with more training experience, while the validation loss curve decreased to a certain value but has begun to increase again. In this case where the validation loss curve increases after reaching its minimum, the model will not perform as efficiently with new data as it is when using its training data.



```
              precision    recall  f1-score   support

        Bean       0.88      0.91      0.89       200
Bitter_Gourd       0.98      0.93      0.95       200
Bottle_Gourd       0.96      0.91      0.93       200
     Brinjal       0.91      0.84      0.88       200
    Broccoli       0.91      0.86      0.89       200
     Cabbage       0.81      0.89      0.85       200
    Capsicum       0.96      0.95      0.96       200
      Carrot       0.93      0.99      0.96       200
 Cauliflower       0.89      0.77      0.83       200
    Cucumber       0.90      0.89      0.89       200
      Papaya       0.93      0.96      0.95       200
      Potato       0.93      0.94      0.93       200
     Pumpkin       0.92      0.84      0.88       200
      Radish       0.89      0.99      0.94       200
      Tomato       0.78      0.89      0.83       200

    accuracy                           0.90      3000
   macro avg       0.91      0.90      0.90      3000
weighted avg       0.91      0.90      0.90      3000
```

Figure 8, Classification Report

Conclusion

The neural network works as expected. A precision of 91% was achieved, with a training accuracy of 98% and a testing accuracy of 91%. To improve its efficiency, a smaller learning rate could be considered for future models. Regularization of the model capacity could help the model achieve higher performance accuracy when adding new data to the network by reducing the size of the hidden layers. Other factors such as the kernel size used, the number of total inputs, the choice of the activation functions and dense layer functions may improve the overall performance of the model. The optimizer Adam showed efficient computationally and was appropriate for the purpose of this study, however other optimizers may be explored in future applications. Adding augmentation to the data would allow it to rotate, flip and resize data more easily and can help prevent overfitting, giving the model a better performance. As a conclusion, the model provides higher accuracy when using the training data than it does with new data (validation), training the model with different functions and parameters like the ones mentioned above would help improve its accuracy.

References

[1] M. Israk Ahmed, &amp; Shahriyar Mahmud Mamun. (2021). *Vegetable Image Dataset* [Data set]. Kaggle. https://doi.org/10.34740/KAGGLE/DSV/2965251

[2] Brownlee, J. (2017), *Gentle Introduction to the Adam Optimization Algorithm for Deep Learning.* [Online] Available at: https://machinelearningmastery.com/adam-optimization-algorithm-for-deep-learning/

[3] Jeffkine, (2016), *Backpropagation Convolutional Networks*. [Online] Available at: https://www.jefkine.com/general/2016/09/05/backpropagation-in-convolutional-neural-networks/

[4] Sadiq, R. (2021, CNN BackPropagation Fall 2021. [Online] Available at: https://deeplearning.cs.cmu.edu/F21/document/recitation/Recitation5/CNN_Backprop_Recitation_5_F21.pdf

[5] Brownlee, J. (2020), *Diagnosis Model Performance with Learning Curves*. [Online] Available at: https://rstudio-conf-2020.github.io/dl-keras-tf/notebooks/learning-curve-diagnostics.nb.html