**Design a Fruit Classification System**

**Abstract:**

*This project presents the creation of a machine learning model used for automatically classification of 14 different types of fruits to help farmers reduce their manual packing and stacking. The dataset used consists of 14 types of fruit images having size 100x100 which are then preprocessed through normalization and training-validation split. Three CNN models were created and trained and one selected with one dropout extra layer with key hyperparameters tuned for optimal performance. The model final test accuracy achieved is 100% with loss score=0.00010261590068694204. Training accuracy and loss curves shows stable learning without significant overfitting. The final model is ready to assist farmers and will offer practical value to agricultural logistics.*

**Introduction:**This report show case the details of a machine learning model that enables a machine to classify fruits just looking on their image labels. In this report we define the whole mythology working behind our model and also see how this suits best to our best options available.

Fruit classification model will help our farmers to sort and automatically pack the fruits and stack them which is useful in their supply chain process and reduce manual labor cost. This model will also increase the quality and can also be enhanced or scaled further to detect ripeness and defects in future.

Here are some key objectives that we are going to achieve in this project:

* Develop a CNN model to classify 14 types of fruits by looking at their images using labels.
* For that we are going to work on image data, preprocess it and make it useful for our model.
* Train the model using a lot of data and make it work on test data accurately.
* Optimize the model performance by tuning different hyperparameters.
* And then we visualize the final results to see the performance.
* Finally, will test it using unseen data to machine.

**Methodology:**

The methodology behind the whole pipeline involves different steps that are explained below:

1. **Data Preprocessing:**

First of all, we need to load the available data into the system. Since it consists of two folders one is test and other is train each having 14 folders each. Those 14 folders also represent labels for each class like apple folder is containing only pictures of apple and vice versa. Since the labels are already present so it’s a type of supervised learning problem. Data is loaded using “Tensorflow/Keras” library using its “image\_dataset\_from\_directory” function. Which not only loads data into model but also split it into train, test and validation dataset. We splited train dataset into 85% training dataset and 15% validation dataset to ensure accurate training before testing it. Here is the brief summary of our data;

* Found 6024 files belonging to 14 classes.
* Using 5121 files for training.
* Found 6024 files belonging to 14 classes.
* Using 903 files for validation.
* Found 663 files belonging to 14 classes.

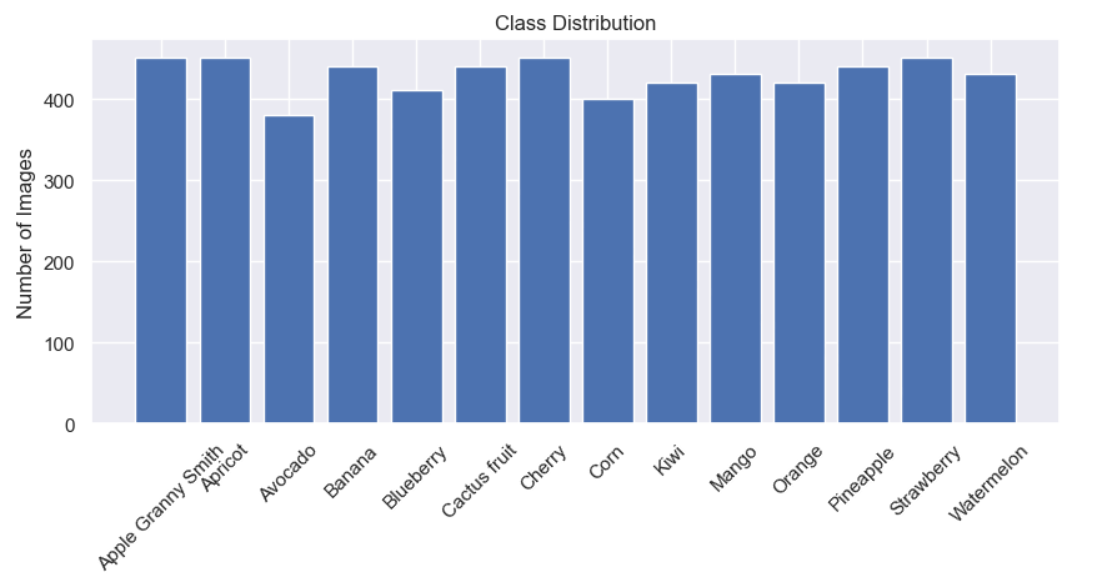
Aspart of preprocessing image data we normally resize the images to ensure same size for each image but in our case they are already of same size which is 100x100 so we are using them as it is.

Another step in data preprocessing is to normalize the data, as part of normalization in image data the pixels of images are scaled to ensure the convergence and training stability.

In our case, after looking into pixel values for each image the maximum pixel number was 255 so we divided all the pixels with 255 to scale them between 0 to 1.

We also need to know the class distribution of our data so that we can prevent our model to behave in a biased manner towards some classes.

Here is the graph fetched out from data that show the class distribution. It looks almost a uniform distribution. But still, we will take some steps to remove any bias.



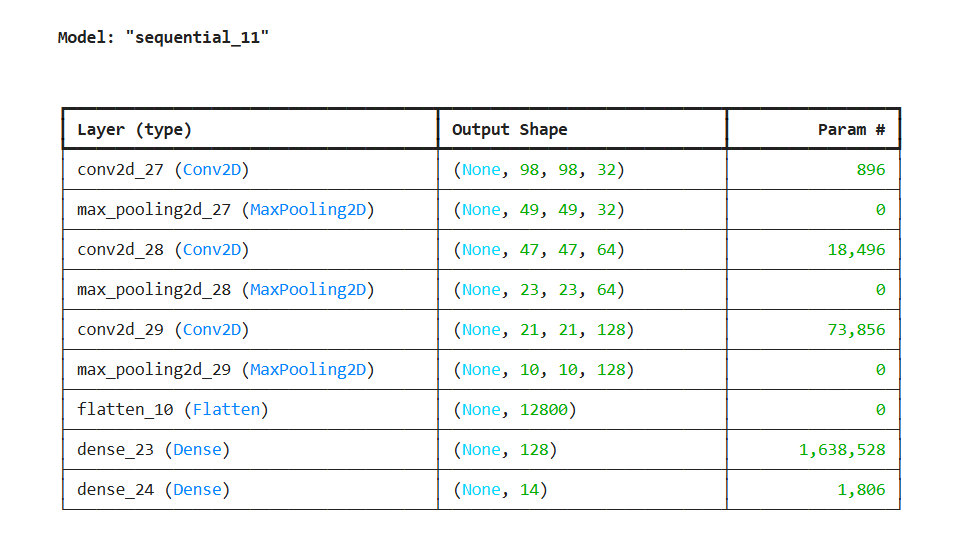
That was all preprocessing done in our case, now our data is suitable for further training and processing.

1. **Model Desing:**

I have created three different models and evaluated their performance by applying them differently. From all three one having a dropout layer was performing much better as it allows the model to remove any overfitting nature which is good in practical. So, I will explain only that one over here.

Since our task follows a linear stack of layers, where each of our layer is following directly into others without any branching logic hence sequential model is pretty good for that kind of task.

So, our model consists of straightforward architecture having details shown below:

****

**Total parameters: 5,200,748 (19.84 MB)**

**Trainable parameters: 1,733,582 (6.61 MB)**

**Non-trainable parameters: 0 (0.00 B)**

**Optimizer parameters: 3,467,166 (13.23 MB)**

So, there was one input layer having shape(100x100) with 3 channels for colored images, 3 convolution layers having same (3x3) size filters (first layer filters=32, second layer filters=64, third layer filters=128), along with 3 pooling layers each having filter size(2x2), then one fully connected flatten layer and at last output dense layer having 14 neurons showing each class probability. There is also another dropout layer added before output dense layer to prevent model overfitting by dropping 50% neurons randomly in each training step.

I used ‘’relu’’ as an activation function in hidden layers to introduce non linearity and complex pattern efficiency in the model and “softmax” in final output layer to get the final results in probability. This model was made using tensorflow library.

1. **Training of model:**

After model building, we need to train our model on train dataset by initializing the hyper parameters and then tunning them to gain the maximum accuracy. Here is the detail of our initial parameters:

Batch size=128

Training dataset=85% of initial train dataset available

Validation dataset=15% of train dataset

No of epochs=10

Learning Rate = 0.001

Optimizer= SGD (Stochastic Gradient Descent)

Loss function='sparse\_categorical\_crossentropy'

Activation Function=relu and softmax

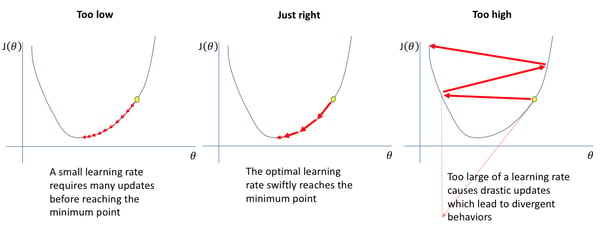
These were our initial selected parameters. Let me discuss a brief usage of each first and then I will tune them.

Batch size: It is the division of whole dataset in subsets each called one batch like in our case 5121 were the total no of training images there so each batch contains 128 images hence 41 total batches were in that case.

Validation Dataset: This is a part of training dataset that model does not use in training purpose but uses it for evaluation purpose to fine-tune hyperparameters before the model is tested. So we are using 15% of training data for this purpose.

No of epochs: It is the number which defines how many times the model sees the entire data during training process. So, 10 here means that our built model is going to process the training data 10 times and each time it updates its parameters to achieve the best performance of model.

Learning Rate: It defines the size of step taken to update parameters while training to minimize the loss score. It controls how fast model is adapted to the problem. The following graph shows how it behaves:



Optimizer: It is used to adjust the parameters of NNs to reduce the loss score. Its objective is to reach global minima. We started with SGD and will see its counterparts upon tunning.

Loss Function: Just calculate the difference between the predicted and true value. We can say it calculate the error in the model. And depending upon that value model parameters are updated.

Activation Function: Used to introduce non linearity in model. It controls the injection of neurons.

1. **Hyperparameter tuning:**

The parameters that cannot be learned directly from model training are hyperparameters. These are tunned manually to achieve best model performance. They can also specify model speed and complexity. The goal of hyperparameter tuning is to find values of weights and biases that lead to best performance under that specific task.

Here in our model, I performed some tuning whose details are mentioned below:

**Hyperparameters:**

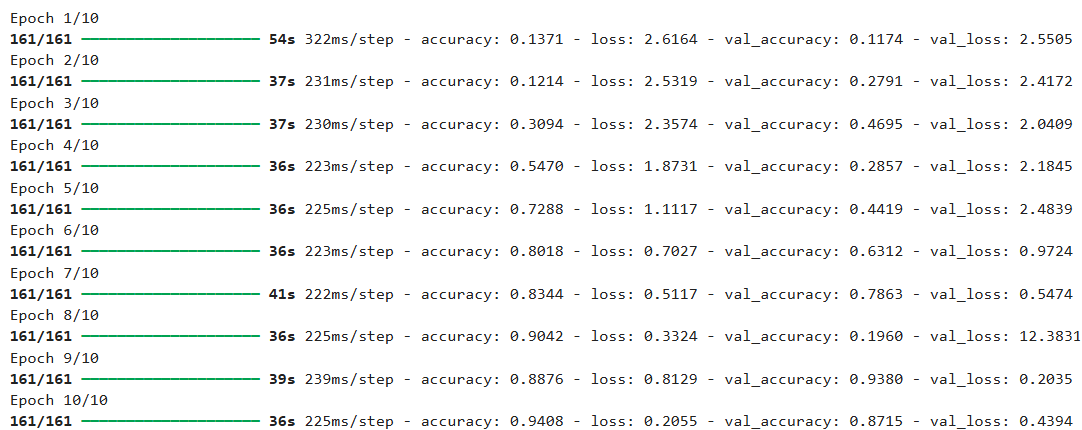
batch\_size=32, Optimizer= SGD(Stochastic Gradient Descent)

validation\_split=0.15 Loss function='sparse\_categorical\_crossentropy'

No of epochs=10 learning\_rate = 0.001

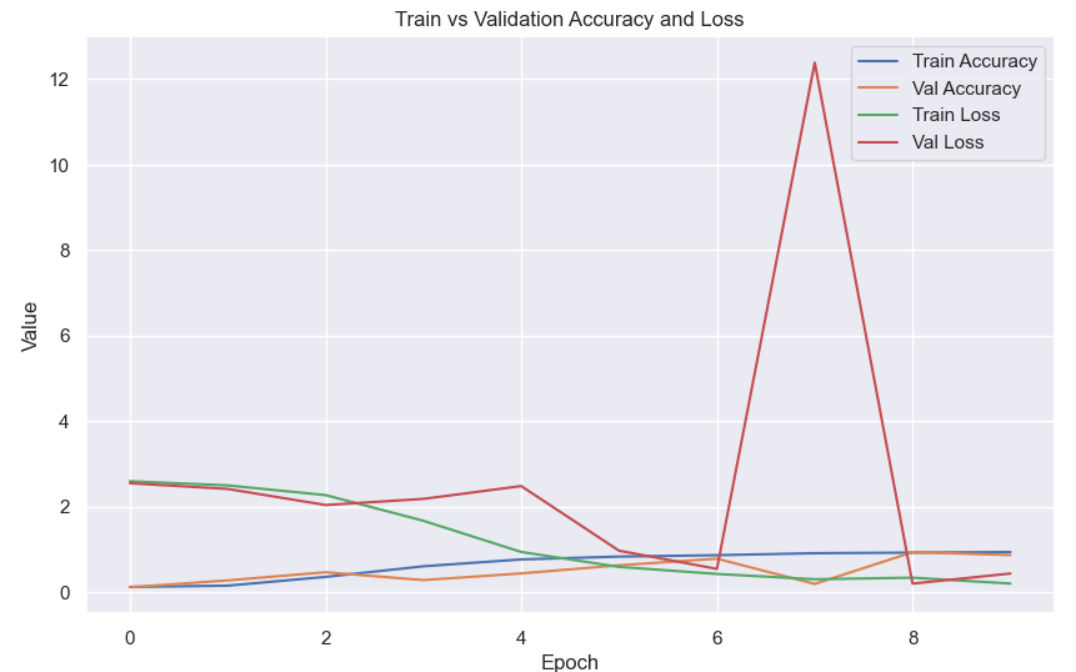
Activation Function=relu and softmax

**Results:**



Final loss value = 0.42790210247039795 Accuracy of the model = 0.8642534017562866

**Accuracy and Loss graph:**



**Analysis:** The sharp spike in validation loss at epoch 7 suggest an anomaly showing overfitting started to appear. Test dataset accuracy is 86% good but can be improved.

**Hyperparameters:**

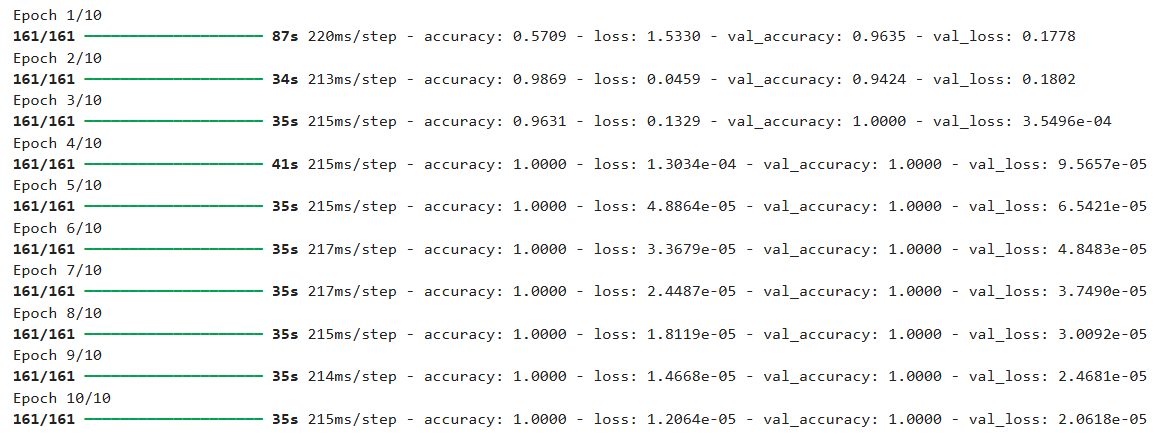
batch\_size=32, *Optimizer= ADAM*

validation\_split=0.15 Loss function='sparse\_categorical\_crossentropy'

No of epochs=10 *learning\_rate = 0.01*

Activation Function=relu and softmax

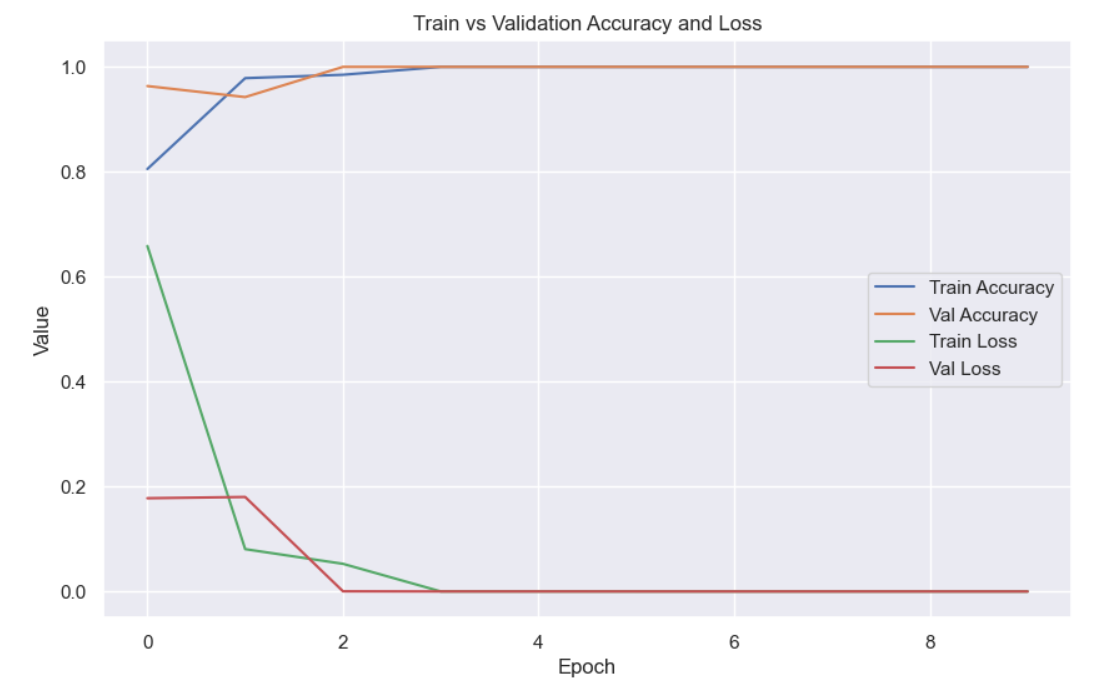
**Results:**



Final loss value = 0.00019415811402723193

Accuracy of the model = 1.0

**Accuracy and Loss Graph:**



**Analysis:** Rapid drop in both losses in first 3 epochs shows model is fitting with data quickly. With 100% accuracy for test data better than last tunning.

**Hyperparameters:**

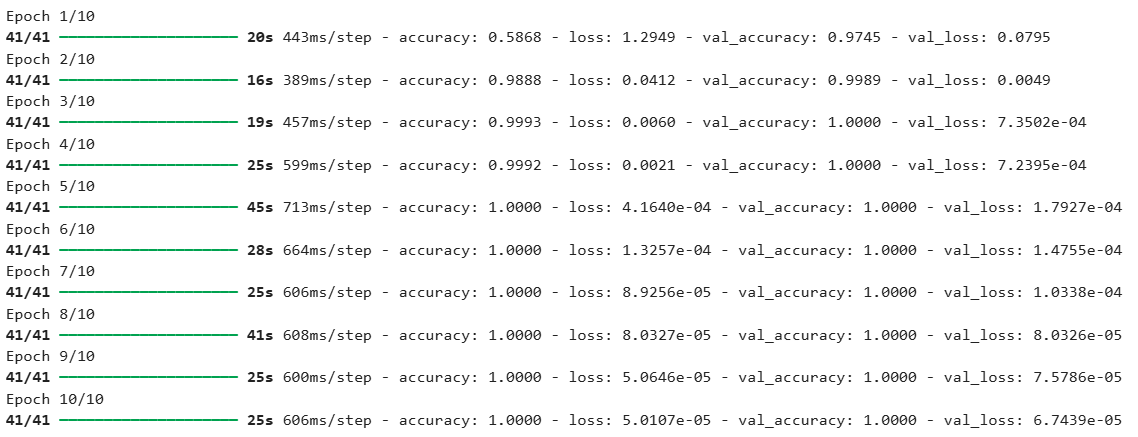
*batch\_size=128,* *Optimizer= ADAM*

validation\_split=0.15 Loss function='sparse\_categorical\_crossentropy'

No of epochs=10 *learning\_rate = 0.001*

Activation Function=relu and softmax

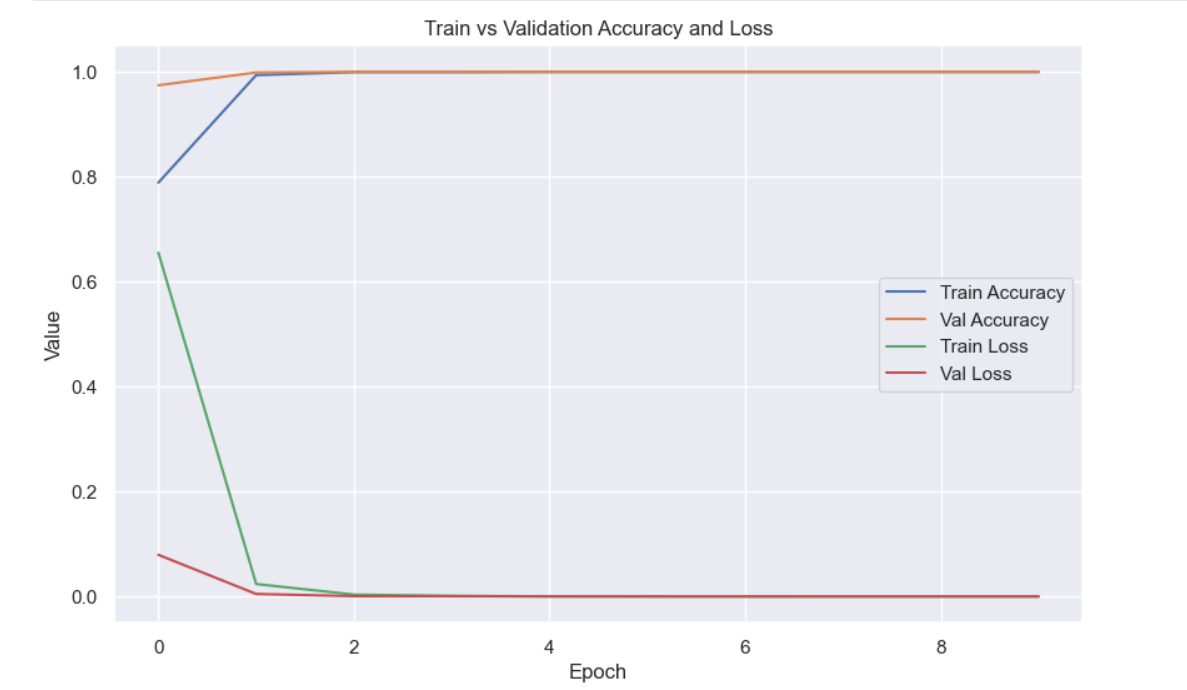
**Results:**



Final loss value = 0.009570944122970104

Accuracy of the model = 0.9984917044639587

**Accuracy and Loss graph:**



**Analysis:** Loss approaches zero quickly suggests that model is converging fast within first 2 epochs. Final test data accuracy is 99%.

**Hyperparameters:**

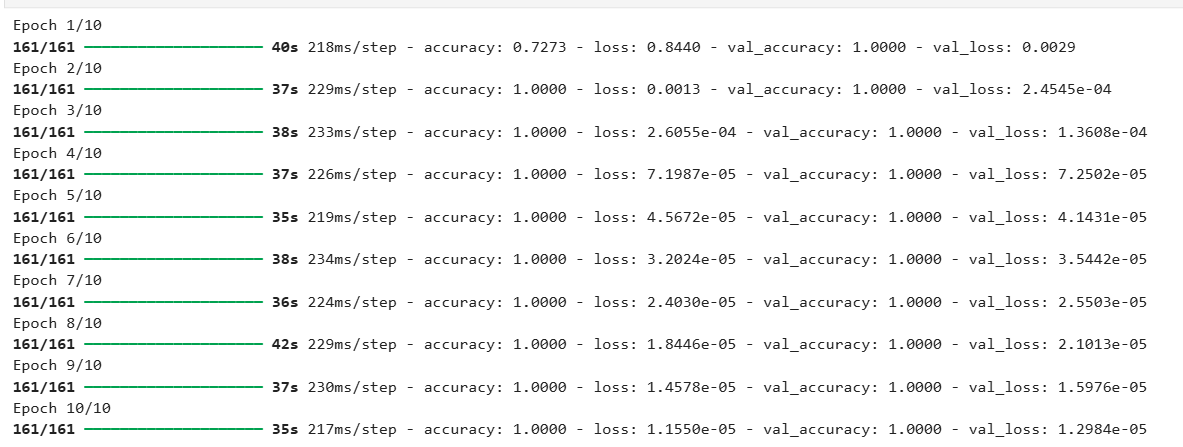
*batch\_size=32,* Optimizer= ADAM

validation\_split=0.15 Loss function='sparse\_categorical\_crossentropy'

No of epochs=10 learning\_rate = 0.001

Activation Function=relu and softmax

**Results:**

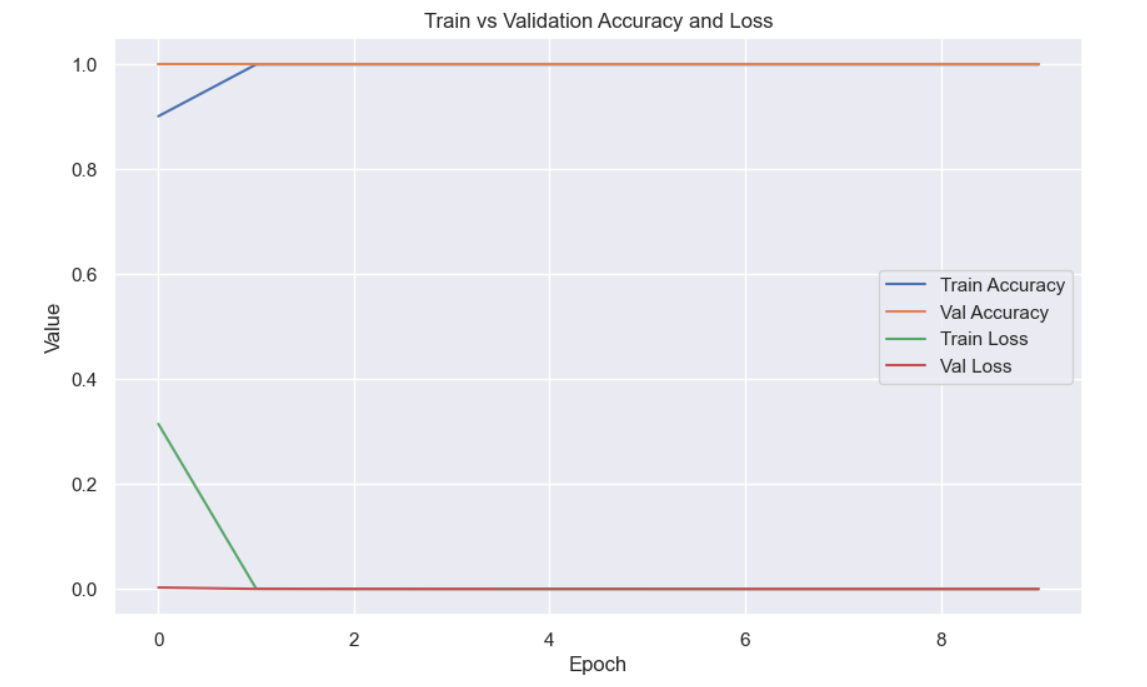


On test Dataset:

Final loss value = 0.004188037943094969

Accuaracy of the model = 0.9984917044639587

**Accuracy and Loss graph:**



**Hyperparameters: With dropout layer.**

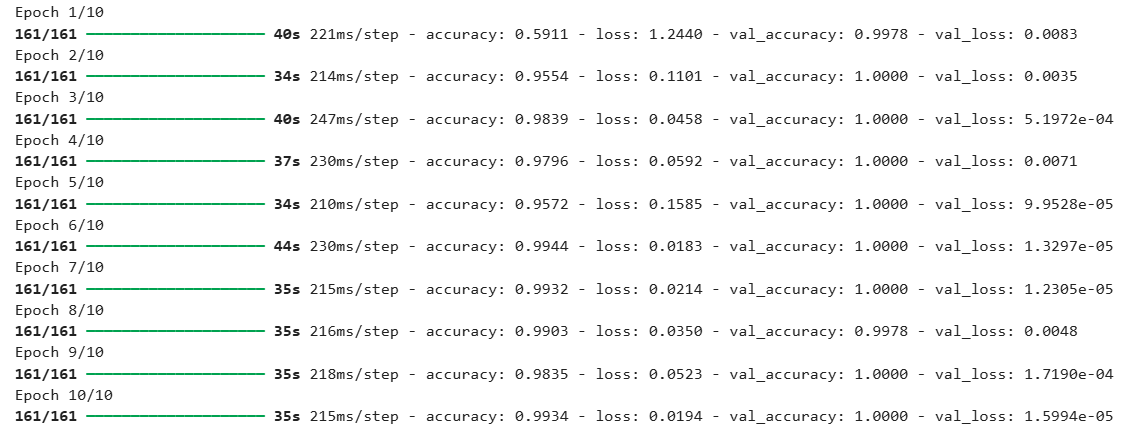
batch\_size=32, Optimizer= ADAM

validation\_split=0.15 Loss function='sparse\_categorical\_crossentropy'

No of epochs=10 learning\_rate = 0.001

Activation Function=relu and softmax With **Dropout layer having value 0.5** or 50% of neurons in that layer are randomly deactivated in each training step.

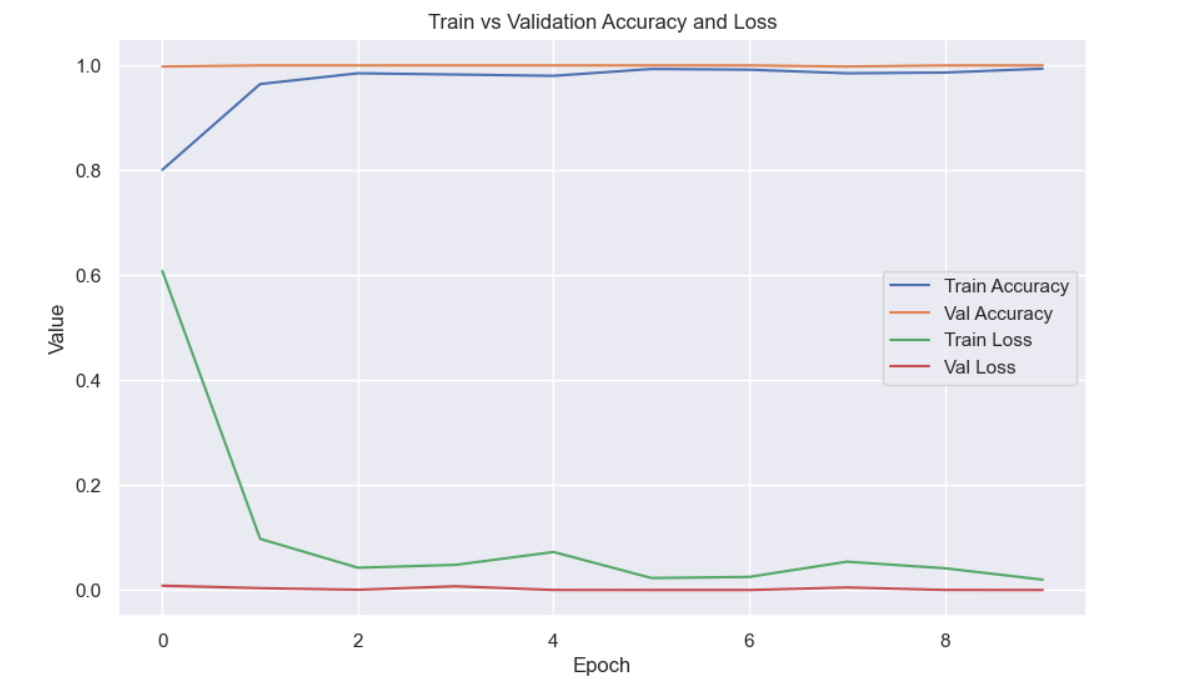
**Results:**



On test dataset:

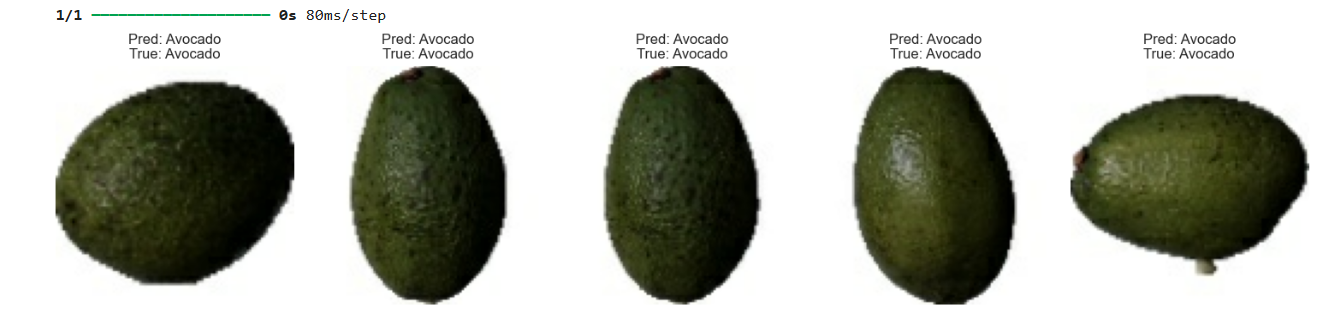
Final loss value = 0.00010261590068694204 Accuracy of the model = 1.0(100%)

**Accuracy and loss graph:**



**Analysis:** Well tuned model which converges quickly and final achived accracy is 100% for test data. This model is perfect among all mentioned above since it perform excellent without just relying on traing data.

**Some test predictions:**





**Conclusion:**

So, our final selected model is the one having an extra dropout layer which helps prevent model to rely to much on training dataset only. And our model relies on more general aspects or features. So now our model is ready to classify any new fruit image and can easily detect the fruit type and stack or pack it in specific category with almost 100% accuracy.