IMAGE CLASSIFICATION OF FRUITS

Dhavalkumar Pithadiya
ID: 22003118

SUMMARY

- Introduction
- Loading The Dataset
- Data Distribution
- Data visualisation
- MobileNetV2 Hypertuning
- Accuracy
- Prediction
- Conclusion
- Coding Link

❑ Introduction:

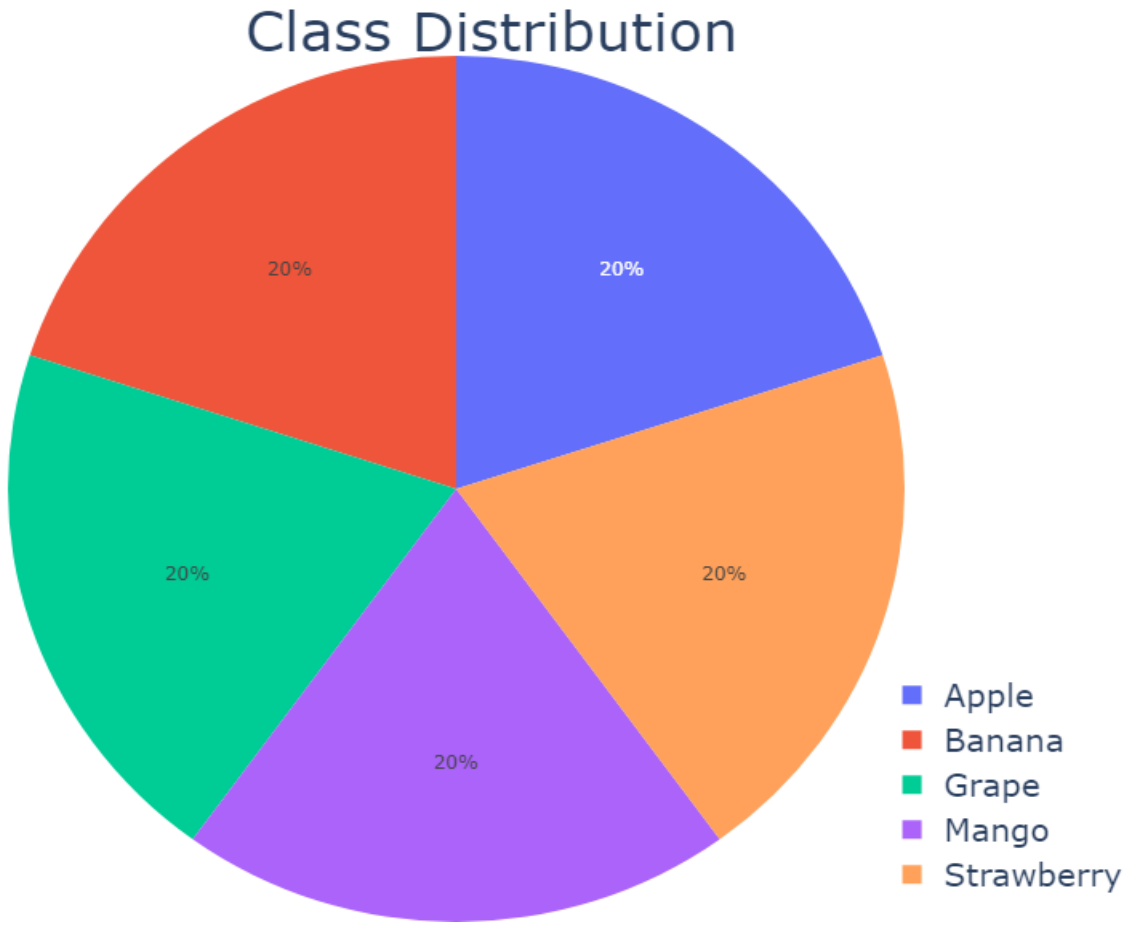
- Here, The given dataset is structured for an image classification task focused on fruits, organized into training, validation, and test sets. The root directory, "Image_Classification," encompasses distinct subdirectories for each set. The dataset comprises a specific number of fruit classes, with class names extracted from the training directory. A random seed is set to ensure reproducibility in both TensorFlow and NumPy.
- The dataset, sourced from Kaggle, is essential for training a fruit image classification model. Kaggle provides diverse datasets, fostering collaborative research and benchmarking within the machine learning community.
- This dataset's real-world applicability enhances the model's ability to generalize to new data, contributing to the dynamic landscape of machine learning development.
- Furthermore, The dataset consists of images belonging to five fruit classes: Apple, Banana, Grape, Mango, and Strawberry. Using given pictures and class name I have build my own model so that can predict the class name for random picture of fruit and that can be helpful in several way in the real world.

❑ Loading The Dataset:

- This code establishes crucial functions for loading and preprocessing a fruit image dataset tailored for classification purposes.
- The load_image function adeptly reads image files, employing tensorflow's I/O functions for flexible decoding of both JPEG and PNG formats. Following this, the image undergoes conversion to a float32 data type and is resized to a standardized dimension.
- The complementary load_dataset function meticulously processes the entire dataset, iterating through specified class names and loading images while accommodating a dataset size limit with the optional trim parameter. Striking a balance between computational efficiency and dataset representativeness, the code incorporates random shuffling to ensure a balanced dataset.
- These meticulously prepared datasets, labeled as x_train, y_train, x_valid, y_valid, x_test, and y_test, lay the groundwork for training and assessing image classification models. The strategic use of tensorflow functions in image handling and preprocessing enhances the efficiency and robustness of subsequent machine learning tasks focused on fruit classification.

❑ Data Distribution:

- Our data is equally distributed in each fruit categories which can be clearly observed in below pie-chart



❑ Data visualisation:

- This function, called show_images, helps us look at pictures from a dataset and their labels. We can decide how many rows and columns of pictures we want to see using parameters like n_rows and n_cols. The function also lets us use a trained machine learning model to make predictions about the pictures.For each row, it randomly picks a few pictures and shows them on the screen. If we provided a model, it even tells us what the model thinks the pictures are.
- This function is handy for checking how well a machine learning model is doing with pictures of fruits, and it's a quick way to see some example images with their labels.
- Here I have showed train Data for all fruits with the class name so we can get data with actual original fruit name.

Apple

Banana

Banana

Strawberry

Apple

Banana

Mango

Banana

Banana

Grape

Grape

Banana

Grape

Mango

Strawberry

MobileNetV2 Hypertuning:

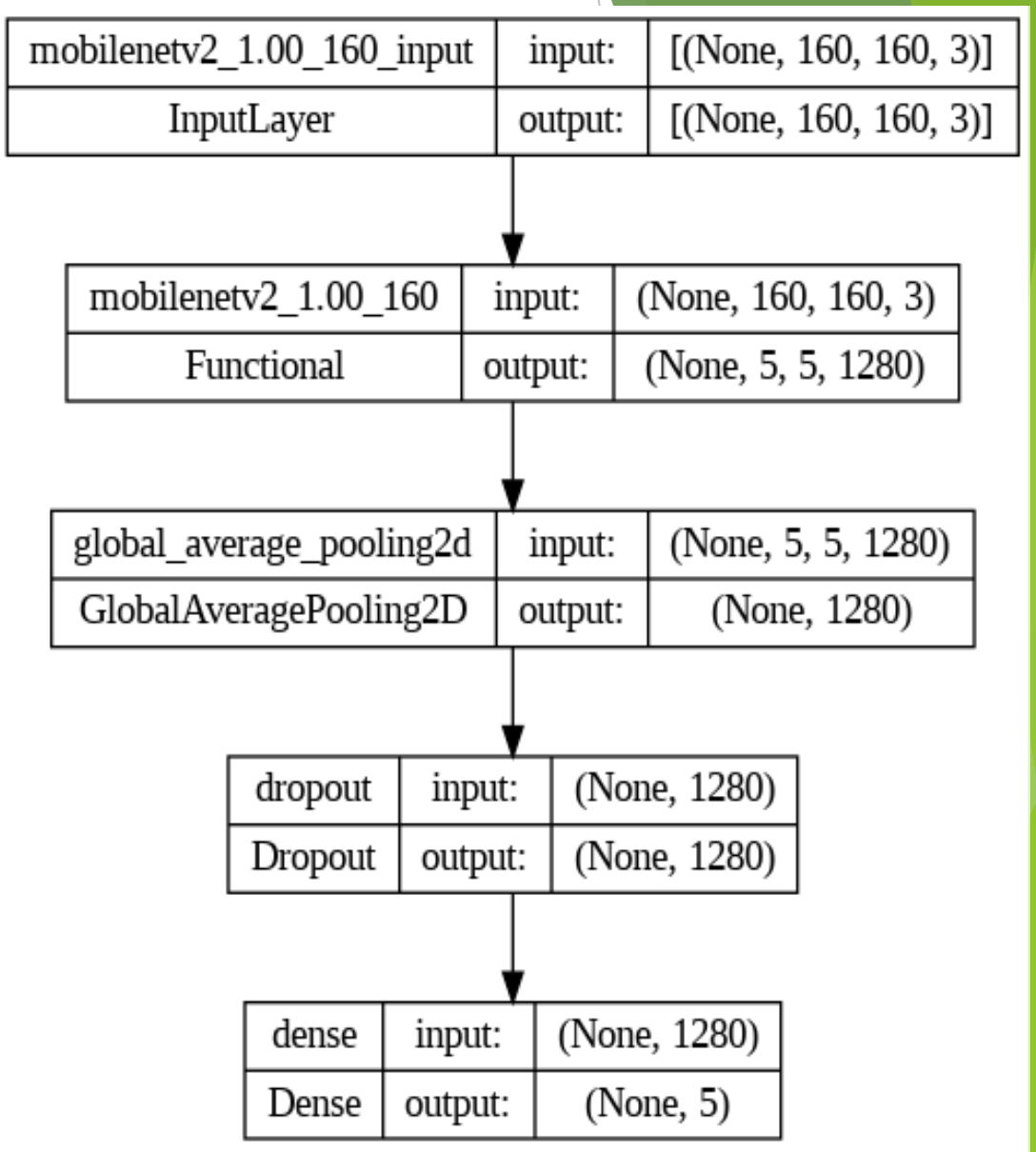
- In this code, a MobileNetV2 backbone is loaded with pre-trained weights from ImageNet, forming the baseline model for an image classification task. The MobileNetV2 architecture is then extended with additional layers, including a global average pooling layer, dropout layer (with a dropout rate of 0.5), and a dense layer with softmax activation to match the number of classes.
- The baseline model is compiled using the categorical crossentropy loss, Adam optimizer with a specified learning rate, and accuracy as the evaluation metric.
- For hypertuning, the baseline model is trained on the training dataset ('X_train', 'y_train') for 20 epochs with early stopping and model checkpoint callbacks. Early stopping monitors the validation loss, stopping the training process if no improvement is observed for a certain number of epochs (patience=3) and restoring the best weights. ModelCheckpoint ensures that only the best model based on validation loss is saved as "MobileNetV2Baseline.h5".
- The batch size for training is set to BATCH_SIZE. After training, the model's performance is evaluated on the test dataset ('X_test', 'y_test'), and the testing loss and accuracy are printed. This hypertuning process aims to enhance the baseline model's performance by finding optimal weights through iterative training epochs while avoiding overfitting, thanks to the early stopping mechanism.

❑ MobileNetV2 Hypertuning:

- We are currently utilizing the MobileNetV2 architecture as our baseline model. This architecture incorporates key components such as a global average pooling layer, a dropout layer with a 50% dropout rate, and a dense layer with softmax activation to classify our data into specific classes represented by n_classes.

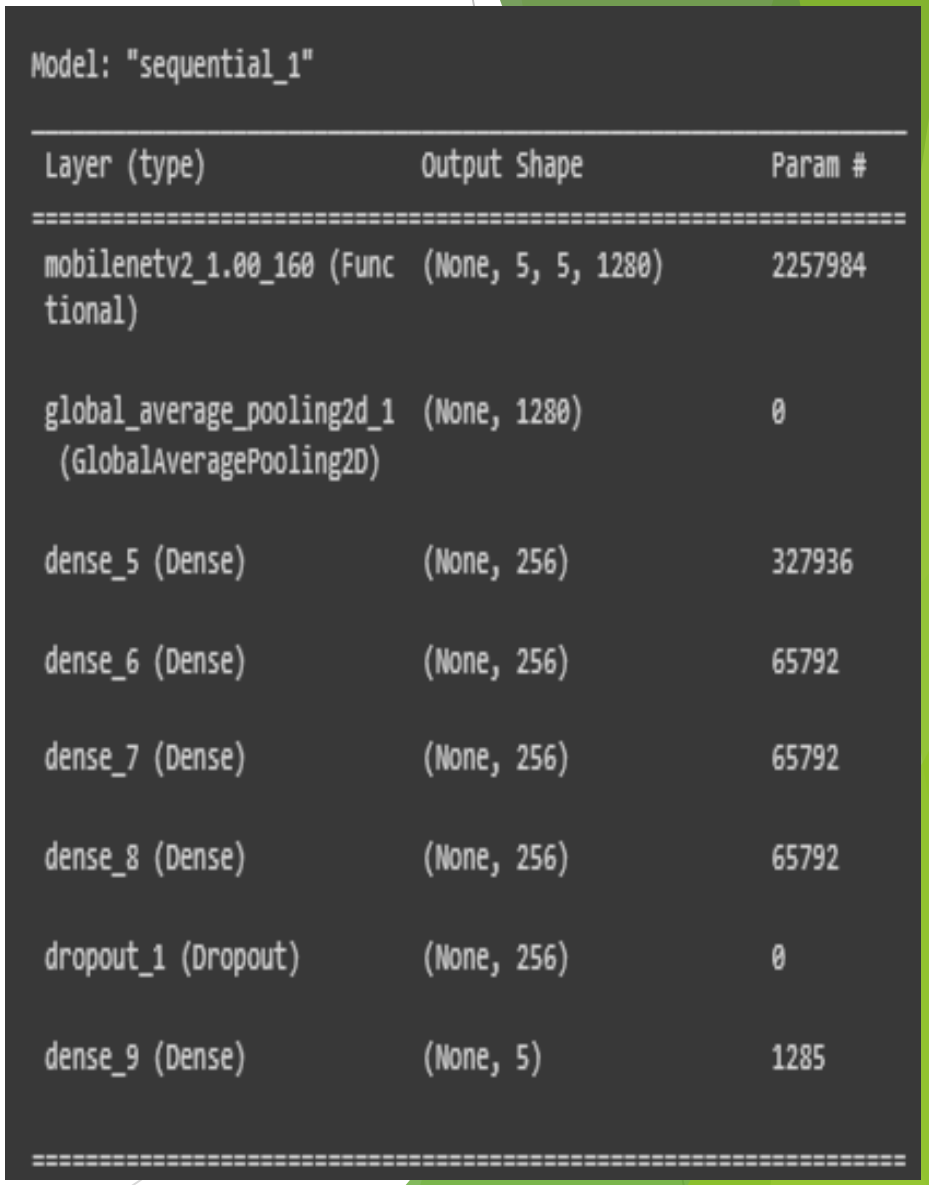
- Upon examining the results, our training accuracy stands at 84%, with the validation accuracy slightly higher at 87%. Notably, the model achieves a peak accuracy of 90%, demonstrating strong performance. More impressively, when tested on unseen data, the model maintains its robustness with a testing accuracy of 92%, suggesting excellent generalization capabilities.

- Given the simplicity of this model architecture, there is potential for improvement through exploration of more complex architectures. Our objective is to discover the optimal model configuration for MobileNetV2 by leveraging hyperparameter tuning with Keras Tuner.



❑ MobileNetV2 Hypertuning:

- To achieve this, we have opted for the random search method provided by Keras Tuner. This method allows us to systematically explore various combinations of hyperparameters, aiding us in identifying the architecture that delivers the highest accuracy and robustness for our specific dataset.
- With the aim of enhancing our model's performance, we embark on a journey to uncover the most effective and efficient architecture for MobileNetV2 using Keras Tuner.
- Here I have created another build_model.
- Then I add more parameters such as hyperparameters() and 20 epochs and build _model for the purpose of strong model and get more accuracy for picture
- Looking forward, As a picture of model summery, we can observe that each layers listed with its types and the output shape .The last column of the picture (param#) showcase the number of parameters for each layers. Which includes trainable and non-trainable parameters.
- Overall, after analysing the picture and above information we can get overview of the model architecture, parameters and their shapes.



```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 mobilenetv2_1.00_160 (Func  (None, 5, 5, 1280)        2257984
 tional)

 global_average_pooling2d_1  (None, 1280)              0
  (GlobalAveragePooling2D)

 dense_5 (Dense)             (None, 256)               327936

 dense_6 (Dense)             (None, 256)               65792

 dense_7 (Dense)             (None, 256)               65792

 dense_8 (Dense)             (None, 256)               65792

 dropout_1 (Dropout)         (None, 256)               0

 dense_9 (Dense)             (None, 5)                 1285

=================================================================
```

❑ Accuracy:

- The graph illustrates the training and validation performance of the MobileNet model. In the Loss Curve, decreasing training and validation losses indicate effective learning, with the dashed green line showcasing good generalization on testing data. The Accuracy Curve depicts improving training and validation accuracies, with the dotted green line signifying strong performance on unseen data. The use of a dark background, sophisticated color palette, and professional font styling enhances the visual impact, making it a concise yet informative representation of the model's training dynamics and generalization capabilities.

Test Accuracy is ~85%

❑ Prediction:

- The model evaluation compares the baseline and optimized MobileNet models using metrics like Precision, Recall, and F1 Score on the test dataset. The optimized model demonstrates improved performance across these metrics, showcasing enhanced classification accuracy. Additionally, a visual display of test images with reduced label font size (set at 10) provides a concise overview of the model's predictions, offering insights into its recognition capabilities for different fruit categories.

- As we can see majority prediction Accuracy are more than 90%
- If we add more epoch into our data than we can generate more accurate model for prediction.

❑ Prediction:

Strawberry
Prediction: Strawberry - 0.9991

Apple
Prediction: Apple - 0.9817

Grape
Prediction: Grape - 0.9997

Grape
Prediction: Strawberry - 0.6798

Strawberry
Prediction: Strawberry - 1.0

Banana
Prediction: Banana - 0.9363

Apple
Prediction: Apple - 0.6357

Mango
Prediction: Mango - 0.9001

Strawberry
Prediction: Strawberry - 0.9958

Strawberry
Prediction: Strawberry - 1.0

❑ Conclusion:

- The exploration of fruit image classification involves a well-curated Kaggle dataset, featuring five distinct fruit classes.
- Leveraging the MobileNetV2 architecture as the baseline model, the journey progresses to hyperparameter tuning using the Keras Tuner library, leading to an optimized model with improved precision, recall, and F1 Score. The dataset's balanced class distribution and efficient preprocessing contribute to effective model training. Visualizations, including an appealing pie chart showcasing class distribution and concise representations of test predictions, provide valuable insights.
- Overall, this comprehensive approach yields a robust and finely-tuned MobileNetV2 model, demonstrating heightened accuracy and potential for practical applications requiring precise fruit recognition.

❑ Coding Link:

- For colab coding Click over Here
- For Dataset Click over Here