BUILDING EFFICIENT FRUIT DETECTION MODEL

Minor Project submitted in partial fulfillment of the requirements for the award of the degree of

BACHELOR OF TECHNOLOGY

IN

COMPUTER SCIENCE AND ENGINEERING

Submitted by

Kichaiahgari Manisha Reddy (221710309027)

Vallapu Reddy Saketh Reddy (221710309060)

R Advaith (221710309050)

Saikam Vishwanath Reddy (221710309052)

Under the esteemed guidance of

Dr Arshad Ahmad Khan Mohammad

Assistant Professor

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

HYDERABAD

DECEMBER 2020

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

DECLARATION

We hereby declare that the Minor Project entitled “BUILDING EFFICIENT FRUIT DETECTION MODEL” is an original work done in the Department of Computer Science and Engineering, GITAM School of Technology, GITAM (Deemed to be University) submitted in partial fulfillment of the requirements for the award of the degree of B.Tech. in Computer Science and Engineering. The work has not been submitted to any other college or University for the award of any degree or diploma.

Date: 10-12-2020

Kichaiahgari Manisha Reddy	**Vallapu Reddy Saketh Reddy**
221710309027	**221710309060**
R Advaith	**Saikam Vishwanath Reddy**
221710309050	**221710309052**

DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

GITAM

(Deemed to be University)

CERTIFICATE

This is to certify that Minor Project entitled “BUILDING EFFICIENT FRUIT DETECTION MODEL” is submitted by Kichaiahgari Manisha Reddy (221710309027), Vallapu Reddy Saketh Reddy (221710309060), R Advaith (221710309050), Saikam Vishwanath Reddy (221710309052) in partial fulfillment of the requirements for the award of degree of Bachelor of Technology in Computer Science and Engineering. The Minor Project has been approved as it satisfies the academic requirements.

Dr Arshad Ahmad khan mohammad
Assistant Professor
Dept. of Computer
Science and Engineering

ACKNOWLEDGMENT

Our Minor Project would not have been successful without the help of several people. We would like to thank the personalities who were part of our seminar in numerous ways, those who gave us outstanding support from the birth of the seminar.

We are extremely thankful to our honorable Pro-Vice Chancellor, **Prof N Siva Prasad** for providing necessary infrastructure and resources for the accomplishment of our Minor Project.

We are highly indebted to **Prof N Seetharamaiah**, Principal, School of Technology, for his support during the tenure of the Minor Project.

We are very much obliged to our beloved **Prof S Phani Kumar**, Head of the Department of Computer Science & Engineering for providing the opportunity to undertake this seminar and encouragement in completion of this Minor Project.

We hereby wish to express our deep sense of gratitude to the project coordinator **Dr. S Aparna**, Assistant Professor, Department of Computer Science and Engineering, School of Technology and our **Dr Arshad Ahmad khan mohammad,** Assistant Professor, Department of Computer Science and Engineering, School of Technology for the esteemed guidance, moral support and invaluable advice provided by them for the success of the Minor Project.

We are also thankful to all the staff members of the Computer Science and Engineering department who have cooperated in making our minor project a success. We would like to thank all our parents and friends who extended their help, encouragement and moral support either directly or indirectly in our minor project work.

Sincerely,

Kichaiahgari Manisha Reddy
(221710309027)

Vallapu Reddy Saketh Reddy
(221710309060)

R Advaith
(221710309050)

Saikam Vishwanath Reddy
(221710309052)

TABLE OF CONTENTS

ABSTRACT

In today's world, everything is becoming automated. Businesses are using Artificial Intelligence to make this possible. Large companies have the capacity to use Artificial Intelligence to make their business process quicker and efficient. This project aims to focus on a Fruit selling businesses. This project aims to use various machine learning and Deep learning techniques like, Random Forest and Convolutional Neural Networks to classify different fruits.

The models will primarily focus on extracting various features like fruit shape, color, size, texture etc. feeding it to a Deep Learning or Machine Learning model and predicting the name of the fruit. Moreover, a thorough analysis on the data and the performance of the Machine learning and Deep learning models will also be presented.

LIST OF FIGURES

1. INTRODUCTION

Sourcing skilled agricultural-related work in the agricultural sector (especially cultivation) is one of the most cost-requesting features in that industry. This is due to growing provision figures, such as irrigation power, water systems, agrochemicals, etc. This is pushing homestead projects and the green sector with little net profits to encounter stress. In these challenges, the generation of sustenance also has to meet the developing needs of a total population that is continuously developing, and this poses a fundamental challenge to come.

The automated gathering may provide a potential solution to this problem by reducing job costs (longer perseverance and high repeatability) and increasing the quality of organic goods, mainly fruit. Therefore in recent decades, there has been an increasing excitement for the use of horticultural robots for harvesting soil products. Different research undertakings, such as monitoring and picking, combine the progress of these levels. Be it as it may, the enhancement of an exact organic product identifying system is a critical advance towards truly robotized reaping robotics, as this is the framework of front-end detection prior to consequent control and handling frameworks; it will not be selected if a natural product is not known or seen.

This progression is thrilling due to numerous components, like variations of enlightenment, impediments, as well as the conditions where the natural substance shows a comparable visual presence to the base. An all-around summarized approach that is invariant and vigorous to illumination and variance in perspective and exceedingly discriminatory portrayals of character/features is a prerequisite to defeating these. In comparison, considering a grocery store that perceives distinctive varieties of vegetative and natural goods is a rehashed task in general stores, where the clerk must describe all that determines the cost. For the most part, the uniform identification use stopped this bundled products issue, but when consumers need to select their commodity, they would almost definitely not package it and should be weighted accordingly.

A well-known solution to this problem is to include codes for each category of soil products; that involves prerequisite problems that the retention is humid, causing measurement errors. A little manuscript with pictures and legends is another arrangement. The problem with this arrangement is that it is boring to turn over the leaflet[5]. Food derived from the field recognition system that computerizes the expense of naming and registration is an excellent response to this dilemma.

In the FE point, in order to obtain a feature vector (FV) for each sample/fruit, the color and shape feature method, and Scale, Invariant Feature Transformation (SIFT) is used. Then this work used the K-Nearest Neighborhood classification system in the research stage. The key objective of this manuscript is, along these lines, to classify fruit images by using its features to conduct classification by applying the method of artificial intelligence (AI) by applying the model of machine learning (ML).

A process of image recognition that consists of the following phases is presented: PP stage, FE stage, and testing phases. Form, shading estimation, and SIFT are used in the FE stage to eliminate a component vector that takes any image into consideration. The research stage utilizes the system of KNN classification.

This project also aims to use Random Forest classifier which basically work on decision trees and is also famously known to be an ensemble technique of decision trees. This algorithm majorly works on the idea of increasing the number of decision trees and using a voting classifier we will predict the class which the image falls under.

Along with Random forest we also made a custom Deep Learning Convolutional Neural Network, where the working of these models is a replication of working of the human brain.

2. LITERATURE SURVEY

This section carries out a detailed survey of the different approaches proposed for fruit detection systems to develop successful autonomous management of agriculture. Although the question of the natural product of fruit detection has been discussed by numerous specialists, such as research introduced by various scholars in[4],[5],[6],[7],[8], and[9], the problem of establishing a quick and accurate system for the identification of organic fruit products perseveres[10].

This is because of the presence and appearance of an outstanding variety of fruits in an agricultural climate. These feature fruit variation and appearance, such as characteristics of form, texture, scale, color, surface, and illumination. In addition, the fruit products are incompletely disconnected in most environmental environments and are expected to continuously exhibit differences in lighting (illustration) and shadowing conditions.

The problem of fruit product identification as an image division problem is solved by various techniques introduced by the current state-of-method (i.e., organic product versus foundation). In[7], tapple exploration issues are studied to boost the expectations of harvesting. They created a structure that recognized apples based on their shading and unmistakable pattern of specular reflection.

Additional information, such as the size and shape of the regular apple, was used to either evacuate incorrect identification or to compartment positions that might hold multiple apples. Along with a variety of heuristic approaches, mainly circular and only certain parts were to be known as a place. A system of division for sweet peppers was introduced in [8].

They used a hyperspectral sensor and a six-band sand camera that used a range of highlights, including rudimentary multi-otherworldly material, uniform records of differentiation, as well as entropy-based surface records. Analyses in profoundly controlled glasshouse conditions demonstrated that this methodology created sensibly exact division results. In any case, the creators noticed that it was not sufficiently precise to fabricate a reliable obstruction map.

Using shaping and shading highlights, many fruit product recognition and arrangement frameworks were proposed. Unique natural product images may, in any event, have comparative shading and shape parameters. Several kinds of image research strategies have been linked in previous years to examine farming for recognition and characterization purposes.

A natural product recognition framework that perceives seven organic products was suggested in [11]. First, they characterize natural product images using the KNN calculation based on the organic product's mean shading estimates, shape roundness estimate, zone, and border estimates. The Euclidean separation is used to quantify the separation between the highlights of the dark natural product estimates with the highlights of each organic product class estimates put away to discover its closest organic product class.

Their recognition results have reached an accuracy of up to 90%[11]. In[12], the approach of perceiving foods grown from the ground consolidates numerous highlights and uses a majority fitting classification model for each probability to improve the accuracy of the general order. This methodology orders food produced from the ground depending on shading, surface, what is more, highlights of appearance.

Each component is freely linked to its reasonable arrangement calculation[12] and strengthened. Spatial Domain Analysis, Fourier Descriptor, and Artificial Neural Network were also used to program the organic product to recognize and arrange it. To start with, the recognition of the natural product shape depends on the shape limit and label using the Fourier Descriptor and Spatial Domain Analysis system.

Using shading data acquired during the preparation of the operation using the Artificial Neural Network, identification of the natural product shading is completed. At that point, recognition of organic product shape and shading recognition methods are combined to recognize and arrange the purposes of natural products. This approach is evaluated using apple, banana, and mango images, and results achieved accuracy[13].

The use of a restrictive, arbitrary environment for the almond division was proposed in[9]. They presented a division method considering five classes that use a Sparse Autoencoder to learn using highlights. Within a random field system, these highlights are further used and have appeared to beat past current state-of-the-art techniques. They performed fantastic division execution yet did not identify objects. In addition, they noticed that a noteworthy test exhibited an impediment. Such a methodology is, instinctively, ready to adapt to low dimensions of condition.

Recently, [6] presented a model for detecting tomatoes by initially carrying out shading based division. At that point, shading and shape highlights are utilized to prepare a classification and Regression Trees classification method. This created a division map and assembled associated pixels into locales. Every area is then announced as detection and to lessen the number of Wrong alerts. They prepared a non-classification method utilizing arbitrary backwoods in organized glasshouse conditions.

Pixel-level division methods for article discovery are adopted in most of the previously modeled processes. Most existing approaches have transcendently inspected natural product identification for yield estimation[4],[7]. For natural products in organized glasshouse situations, the limited investigations that have led to accurate fruit identification are carried out as such.

The problem of natural product detection in very testing conditions remains unresolved, all things considered. This is due to the more significant fluctuation in the presence of the objective articles in the setup of the agrarian environment, which implied that the great techniques for sliding window (SW) based methods, despite the fact that indicating great execution when attempted on data sets of chosen images[14], can not cope with the inconstancy in scale and presence of the objective items when transmitted in real.

3. PROBLEM ANALYSIS

The problem statement is to automate the task of identification of different kinds of fruits. In our case there are 5 kinds of fruits namely ‘Apple’, ‘Banana’,’Orange’, ‘Mango’,’JackFruit’. The approach to this problem is using machine learning framework i.e using Algorithms which are based on learning parameters.

As part of exploratory data analysis considering each sample in each of 5 classes initially we need to collect the data and do some preprocessing on the images which is detailed in the further sections of the documentation.

Now the problem is to identify the correct model to train the classification algorithm, as far as the base paper is concerned Pavan Kunchur *et.al.* They implemented K nearest neighbor classification algorithm.

But there are no certain metrics that were presented in the base paper, whereas we in the due course of implementing the project using KNN algorithm analyzed all the metrics which are discussed in the Result Analysis section, with this base paper implementation actually done.

But KNN algorithm doesn’t work very well over clustering the image data to overcome this problem.

We went ahead and implemented an ensemble technique named Random Forest Classifier which in turn resulted in the best accuracy among all the proven algorithms.

We also implemented a Deep learning based Convolutional Neural Network. Which of course may not give the best result as far as accuracy is considered but Convolutional neural Network is a network which works as the human eye.

All the results of the implemented model are present in the result section of the Documentation.

4. DESIGN

4.1 PROJECT DESIGN:

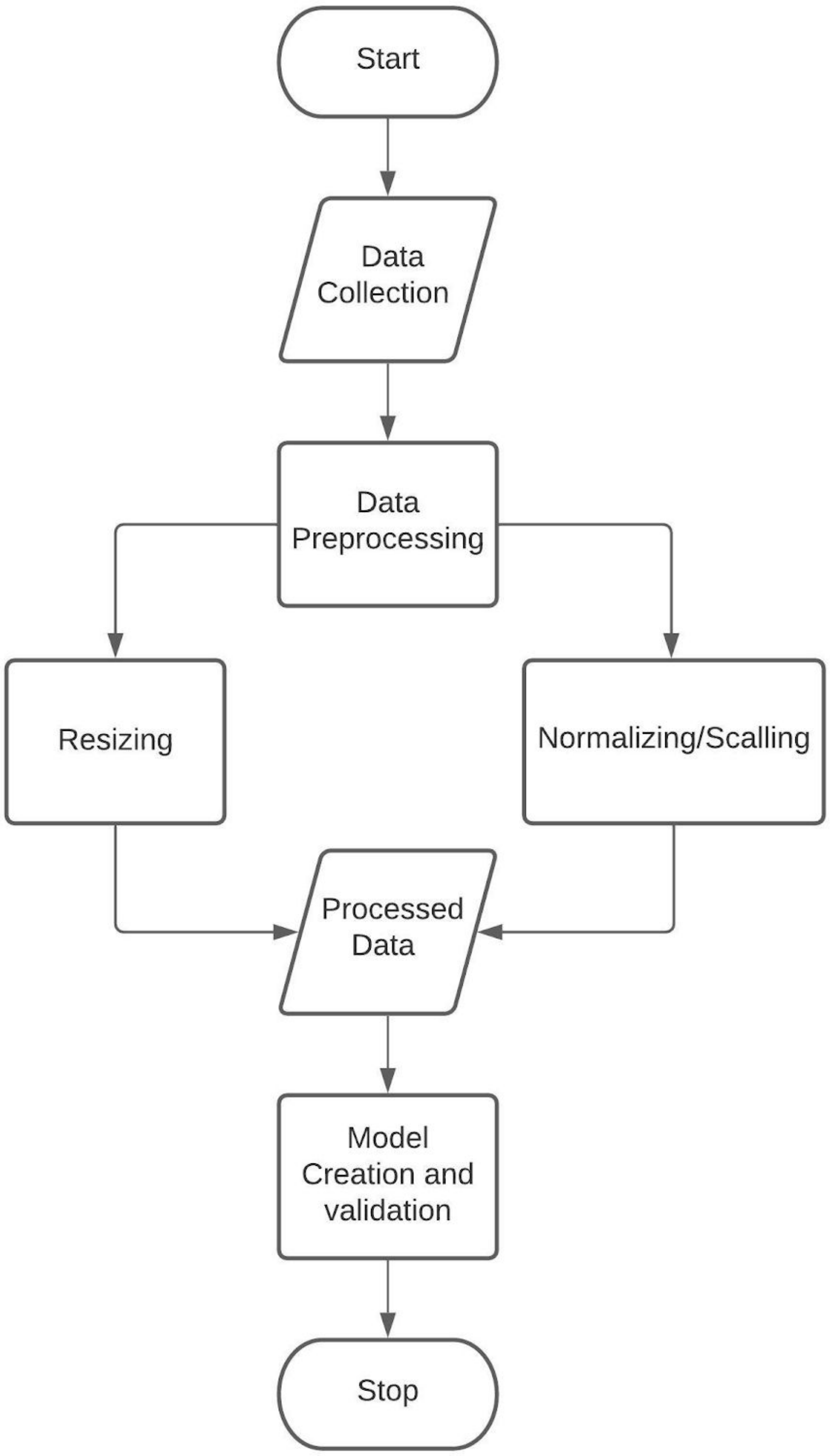


Fig 4.1.1: Project Design

Design of the project flows in the below mentioned phases:

All the phases of the design are detailed in the implementation section of the Documentation.

Phases:

Data collection:

- Initially, we need to collect the images of each class, namely 'Apple,' 'Banana,'' Mango,' 'Orange,' 'JackFruit.'

Data preprocessing:

- After the data collection, we need to preprocess the images, i.e., resizing and normalizing/scaling the image pixels.
- As a part of the resizing phase, initially, we have images of different sizes, which then we have converted images into 90*90 pixels.
- We have scaled the data set in the normalizing/scaling phase by dividing each image pixel with 255 and made each pixel range to be in between 0 to 1.

Model Creation:

- Now we need to select the model to train in this case, We analyzed different models.
- Then we need to train the model over normalized images.
- Evaluate the trained model over different model metrics.

Testing and Validation:

- Evaluate the model over test data.

Performance evaluation:

- Then, we need to evaluate the performance of the model.

4.2 PROJECT ARCHITECTURE:

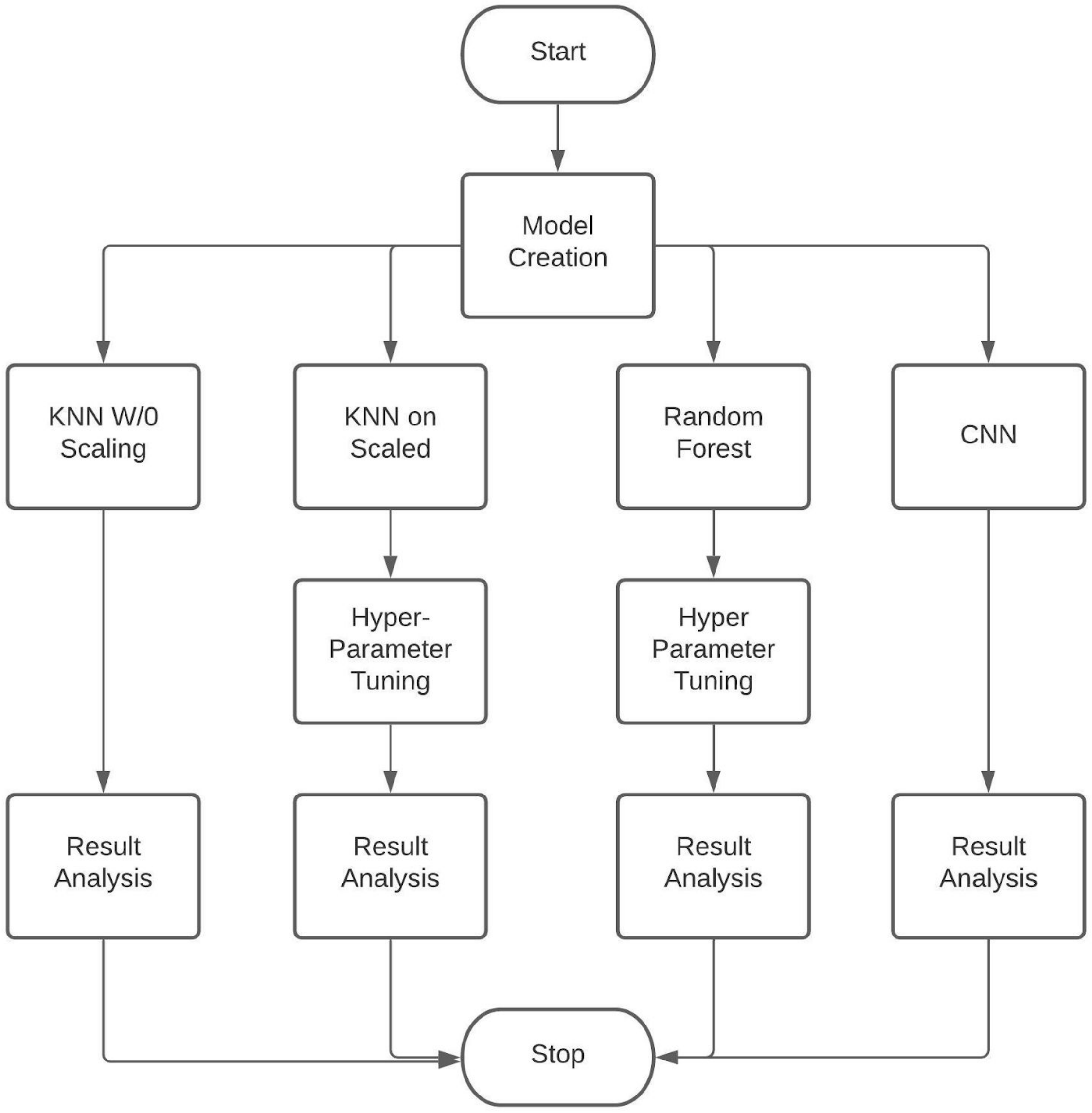


Fig 4.2.1: Project Architecture

The above Figure Project Architecture depicts the flow of the model creation and result evaluation steps as mentioned in the design section we analyzed different machine learning models, they are KNN which is the base algorithm for the project and even this base algorithm is implemented in 2 ways in terms of providing data to it, in the first way we provided the model with unscaled data and analyzed the results the second way is done using providing the scaled data for the model to train.

The two algorithms, Random forest, and Convolutional Neural network are also trained over the same data. To analyze the model more, we gave the models (KNN and Random Forest) for Hyper Parameter Tuning in over to find the best fit parameters. All the results are analyzed and provided in the result section of the project Documentation.

5. IMPLEMENTATION

Implementing the project goes through phases of generic machine learning problem solution.

The steps are:

1. Data Collection
2. Data preprocessing
3. Model Creation
4. Training the model
5. Validating the model
6. Metrics

5.1 DATA COLLECTION:

In this project, data is in the form of images of fruits labeled data and is downloaded from kaggle fruit 360 dataset but this dataset consists of 131 different classes of fruits. It is downloaded from 'https://www.kaggle.com/moltean/fruits' .

After collecting the dataset we separated images of all required classes namely ' apple',' banana', 'jackfruit', 'orange', and 'mango'.

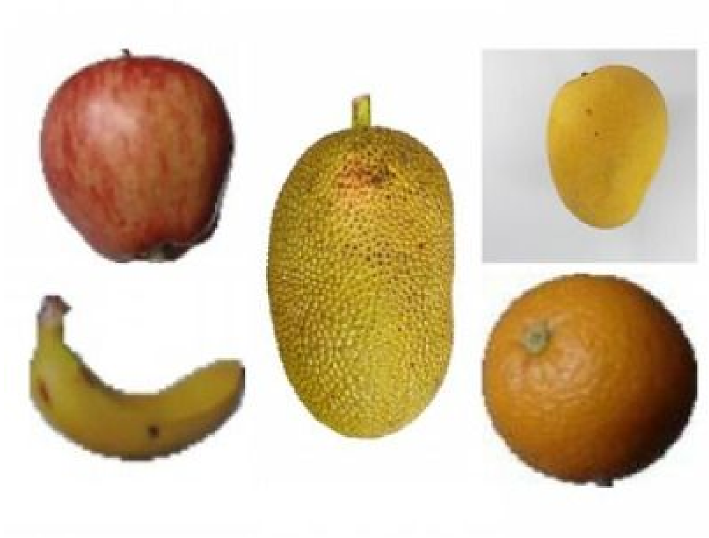


Fig 5.1.1 Images in Dataset

There are 100 images in each label or class:

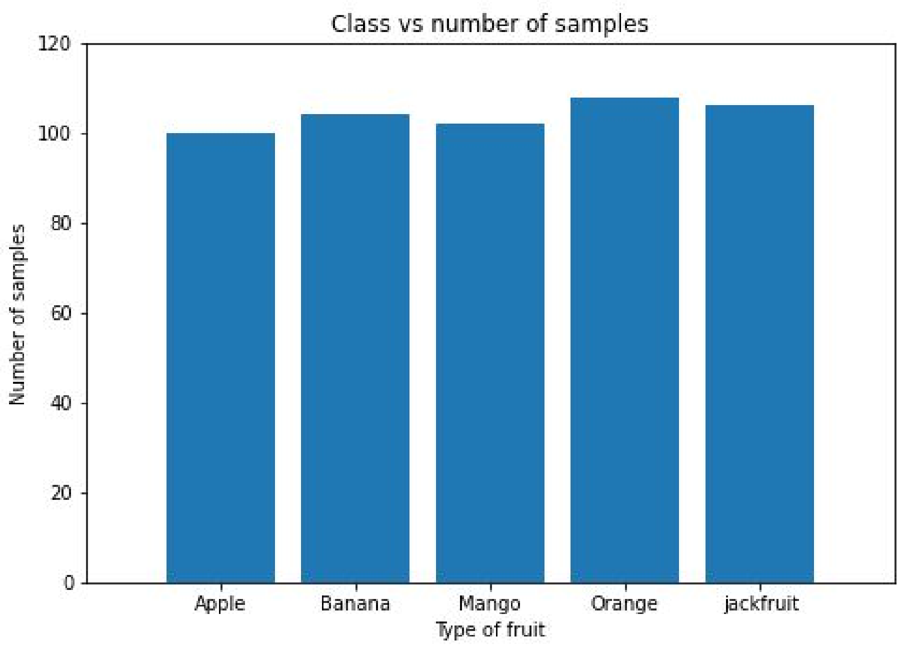
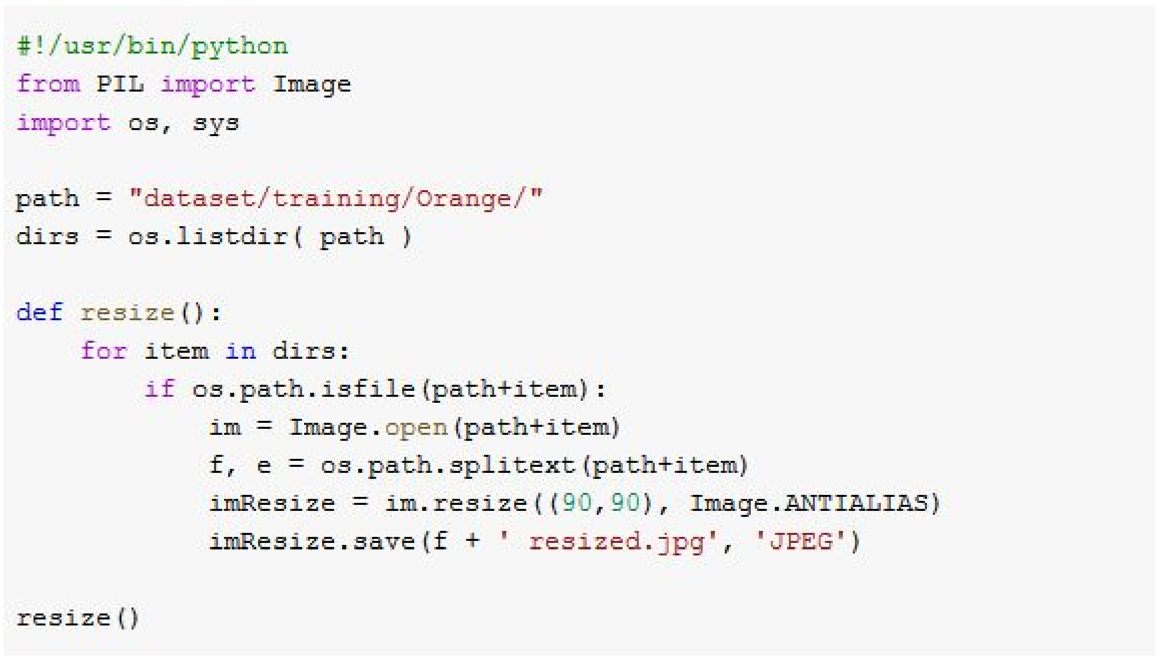


Fig 5.1.2: Number of samples in each class

5.2 DATA PREPROCESSING:

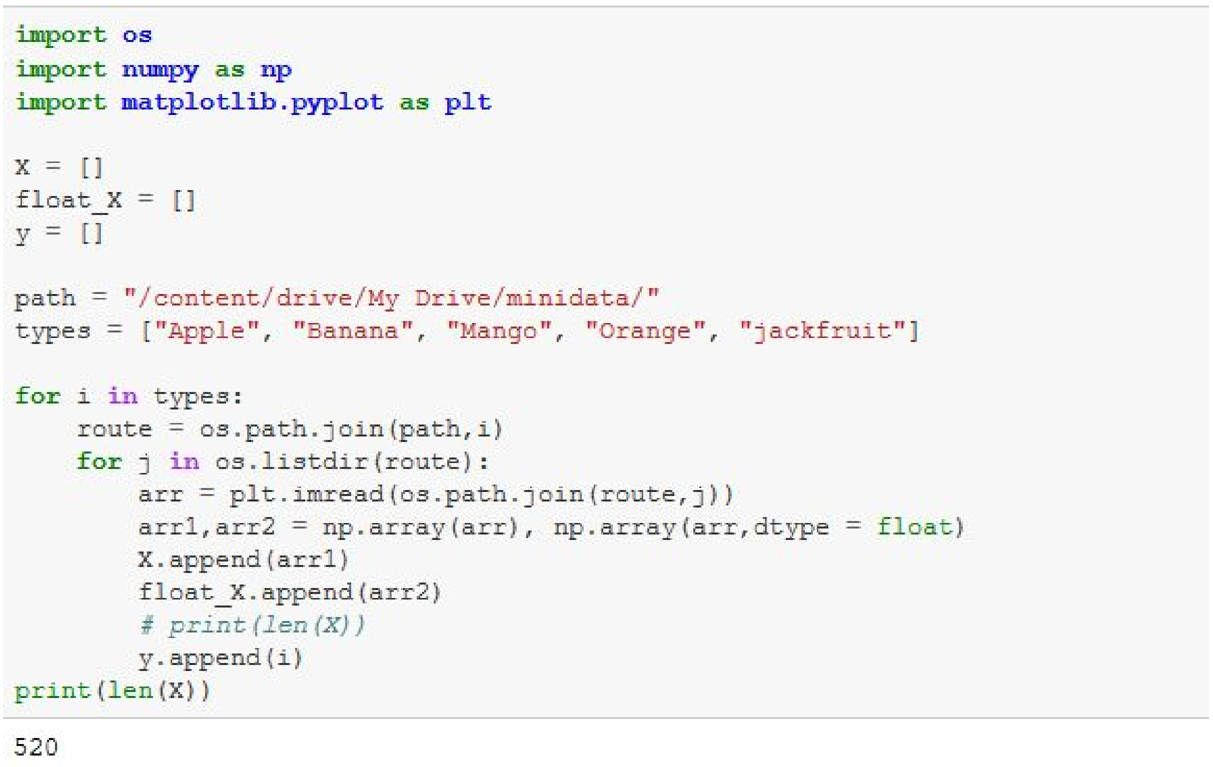
The images in the dataset are irregularly sized so firstly we resize the images to 90 x 90 pixels. To do that we wrote the following code



```
#!/usr/bin/python
from PIL import Image
import os, sys

path = "dataset/training/Orange/"
dirs = os.listdir( path )

def resize():
    for item in dirs:
        if os.path.isfile(path+item):
            im = Image.open(path+item)
            f, e = os.path.splitext(path+item)
            imResize = im.resize((90,90), Image.ANTIALIAS)
            imResize.save(f + ' resized.jpg', 'JPEG')

resize()
```

Fig 5.2.1: Resizing of images

After resizing the image we need to load the images into NumPy arrays in order to give input to our mode.

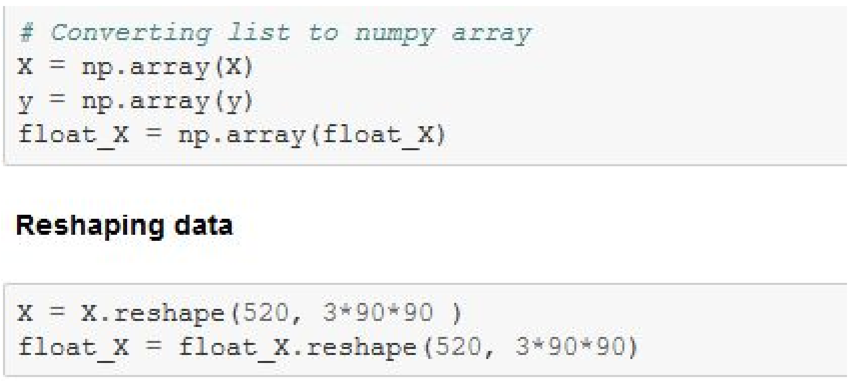


```
import os
import numpy as np
import matplotlib.pyplot as plt

X = []
float_X = []
y = []

path = "/content/drive/My Drive/minidata/"
types = ["Apple", "Banana", "Mango", "Orange", "jackfruit"]

for i in types:
    route = os.path.join(path,i)
    for j in os.listdir(route):
        arr = plt.imread(os.path.join(route,j))
        arr1,arr2 = np.array(arr), np.array(arr,dtype = float)
        X.append(arr1)
        float_X.append(arr2)
        # print(len(X))
        y.append(i)
print(len(X))
```

```
520
```

Fig 5.2.2: Loading Dataset

Then, we reshaped the image from 4D to 2D i.e., (520, 3*90*90).



```
# Converting list to numpy array
X = np.array(X)
y = np.array(y)
float_X = np.array(float_X)
```

Reshaping data

```
X = X.reshape(520, 3*90*90 )
float_X = float_X.reshape(520, 3*90*90)
```

Fig 5.2.3: Reshaping Data

5.3 MODEL CREATION:

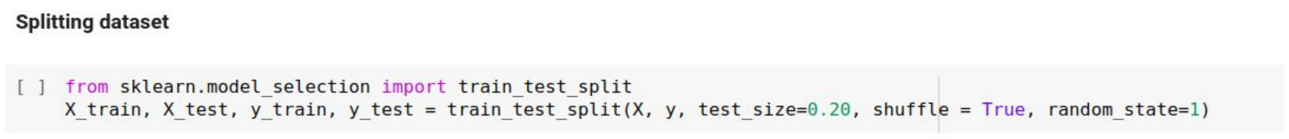
Model creation is a step where we declare model parameters and split the data into train and test sets.

For all these steps we used an open source library called sci-kit learn:

The scikit-learn project started as scikits.learn a Google Summer of Code project by David Cournapeau. Its name stems from the notion that it is a "SciKit" (SciPy Toolkit), a separately-developed and distributed third-party extension to SciPy.[3] The original codebase was later rewritten by other developers. In 2010 Fabian Pedregosa, Gael Varoquaux, Alexandre Gramfort and Vincent Michel, all from the French Institute for Research in Computer Science and Automation in Rocquencourt, France, took leadership of the project and made the first public release on February the 1st 2010.[4] Of the various scikits, scikit-learn, as well as scikit-image, were described as "well-maintained and popular" in November 2012.

Scikit-learn is one of the most popular machine learning libraries on GitHub. Scikit-learn (formerly sci-kit. learn and also known as sklearn) is a free software machine learning library for the Python programming language. It features various classification, regression and clustering algorithms including support vector machines, random forests, gradient boosting, *k*-means, and DBSCAN, and is designed to interoperate with the Python numerical and scientific libraries NumPy and SciPy.

To split the data into training and test sets we use the following code:


Splitting dataset

```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, shuffle = True, random_state=1)
```

Fig 5.3.1: Splitting Dataset

As mentioned in our base paper we are using K nearest neighbour algorithm.

5.3.1 Algorithm (KNN):

The k-nearest neighbors (KNN) algorithm is a simple, easy-to-implement supervised machine learning algorithm that can be used to solve both classification and regression problems.The KNN algorithm assumes that similar things exist in close proximity. In other words, similar things are near to each other.

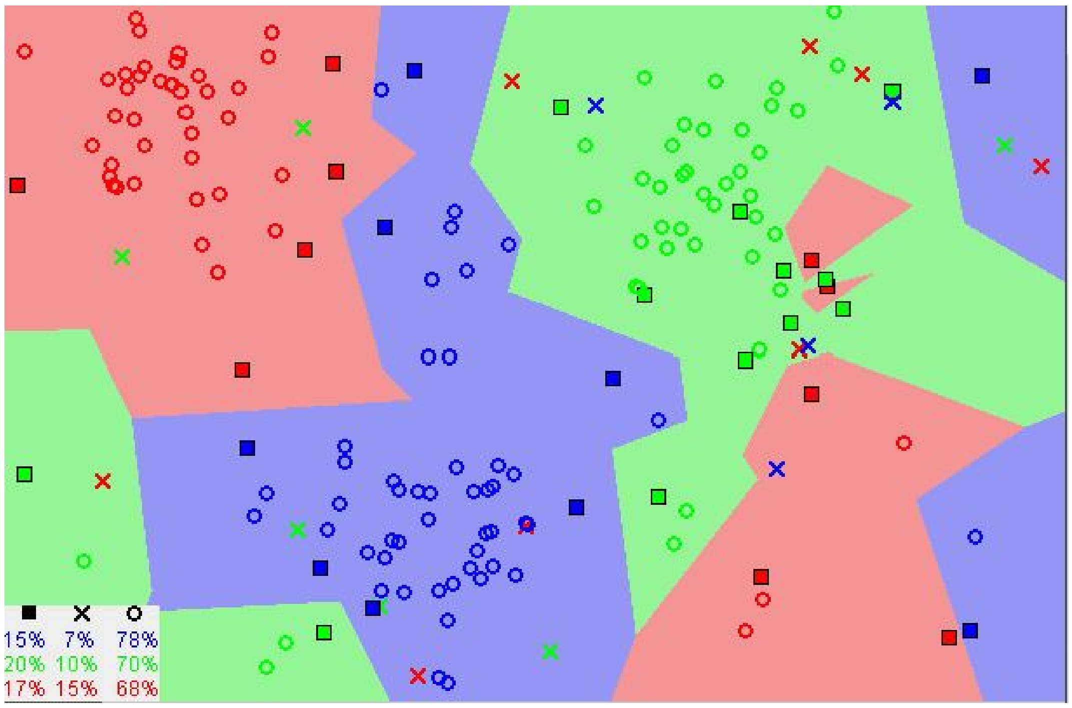


Fig 5.3.1.1: [17] KNN Classification

In the image above, most of the time, similar data points are close to each other. The KNN algorithm hinges on this assumption being true enough for the algorithm to be useful. KNN captures the idea of similarity (sometimes called distance, proximity, or closeness) with some mathematics we might have learned in our childhood— calculating the distance between points on a graph. There are other ways of calculating distance, and one way might be preferable depending on the problem we are solving. However, the straight-line distance (also called the Euclidean distance) is a popular and familiar choice.

(i) Choosing the right value for K:

To select the K that's right for your data, we run the KNN algorithm several times with different values of K and choose the K that reduces the number of errors we encounter while maintaining the algorithm's ability to accurately make predictions when it's given data it hasn't seen before.

Below points are to be taken care of to choose the right k value:

1. As we decrease the value of K to 1, our predictions become less stable. Just think for a minute, imagine K=1 and we have a query point surrounded by several reds and one green (I'm thinking about the top left corner of the colored plot above), but the green

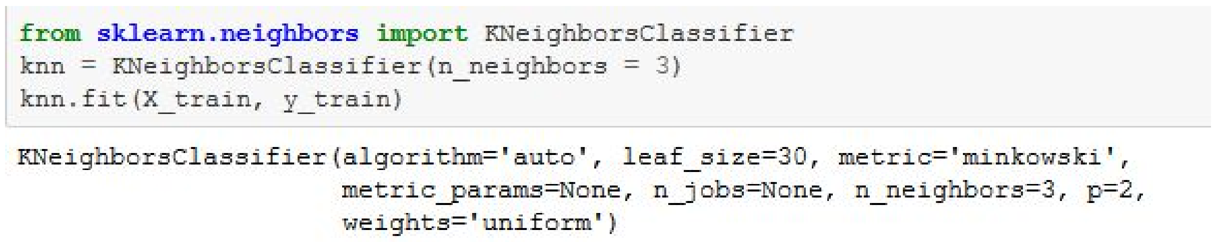
is the single nearest neighbor. Reasonably, we would think the query point is most likely red, but because K=1, KNN incorrectly predicts that the query point is green.

2. Inversely, as we increase the value of K, our predictions become more stable due to majority voting / averaging, and thus, more likely to make more accurate predictions (up to a certain point). Eventually, we begin to witness an increasing number of errors. It is at this point we know we have pushed the value of K too far.
3. In cases where we are taking a majority vote (e.g. picking the mode in a classification problem) among labels, we usually make K an odd number to have a tiebreaker.

Advantages of using KNN:

1. The algorithm is simple and easy to implement.
2. There's no need to build a model, tune several parameters, or make additional assumptions.
3. The algorithm is versatile. It can be used for classification, regression, and search

Implementation of KNN:



```
from sklearn.neighbors import KNeighborsClassifier
knn = KNeighborsClassifier(n_neighbors = 3)
knn.fit(X_train, y_train)
```

```
KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                     metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                     weights='uniform')
```

Fig 5.3.1.2: KNN implementation W/o Scaling

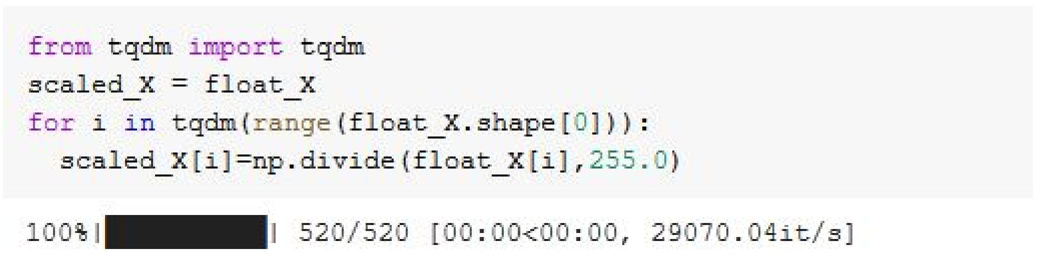
The above code screenshot is the implementation of KNN using the scikit-learn open source module.
Initially, we used default parameters.

They are:

1. Algorithm 'auto': there are different kinds of algorithms where k will converge the data into clusters they are
   a. Balltree: It for fast generalized N-point problems
   b. Kd_tree: KDTree for fast generalized N-point problems
   c. Brute: 'brute' will use a brute-force search.

   d. auto:‘auto’ will attempt to decide the most appropriate algorithm based on the values passed to fit method.
2. Leafsize(30): Leaf size passed to BallTree or KDTree. This can affect the speed of the construction and query, as well as the memory required to store the tree. The optimal value depends on the nature of the problem.
3. Metric (Minkowski): the distance metric to use for the tree. The default metric is Minkowski, and with p=2 is equivalent to the standard Euclidean metric. See the documentation of DistanceMetric for a list of available metrics. If the metric is “precomputed”, X is assumed to be a distance matrix and must be square during the fit. X may be a sparse graph, in which case only “nonzero” elements may be considered neighbors.
4. N_neighbours: Number of neighbors to use by default for kneighbors queries.

To make data more scaled or make all pixels values to range in between 0 to 1 so that data will be organized and there will not be any irregularities in the pixel values.

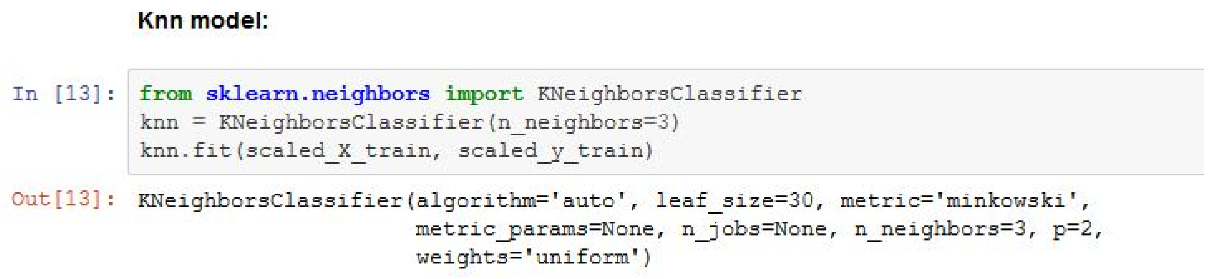


```
from tqdm import tqdm
scaled_X = float_X
for i in tqdm(range(float_X.shape[0])):
  scaled_X[i]=np.divide(float_X[i],255.0)
```

```
100%|██████████| 520/520 [00:00<00:00, 29070.04it/s]
```

Fig 5.3.1.3: Scaling Data

After scaling the data the scaled data is again trained using KNN.



Knn model:

```
In [13]: from sklearn.neighbors import KNeighborsClassifier
         knn = KNeighborsClassifier(n_neighbors=3)
         knn.fit(scaled_X_train, scaled_y_train)

Out[13]: KNeighborsClassifier(algorithm='auto', leaf_size=30, metric='minkowski',
                              metric_params=None, n_jobs=None, n_neighbors=3, p=2,
                              weights='uniform')
```

Fig 5.3.1.4: KNN Implementation on Scaled Data

In order to find the best k value we need to use a technique called hyperparameter tuning, for this project we used GridSearchCV.

Hyper parameterTuning:

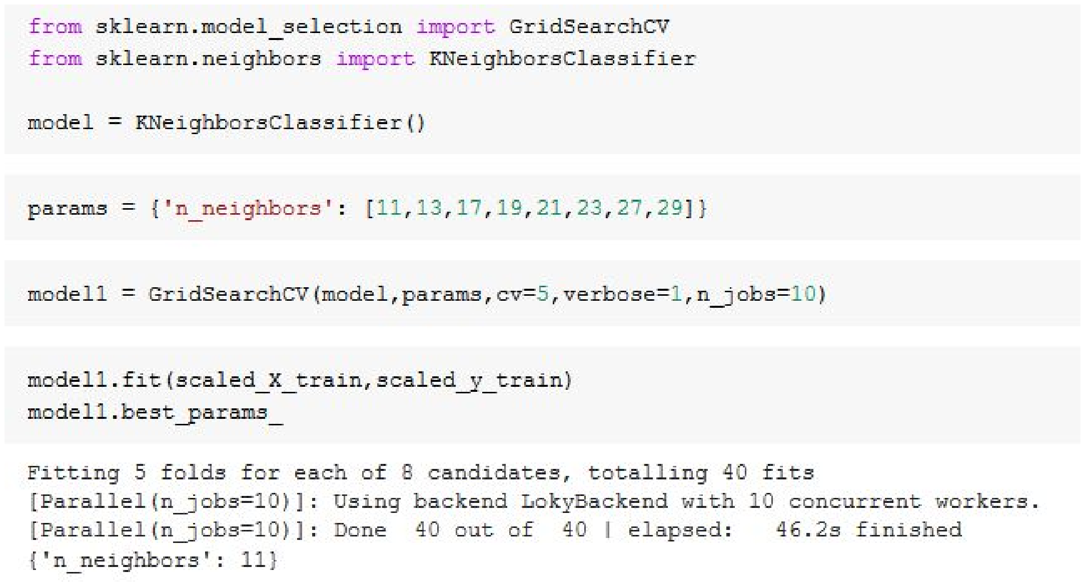
Parameters which define the model architecture are referred to as hyperparameters and thus this process of searching for the ideal model architecture is referred to as hyperparameter tuning.

Hyperparameters are not model parameters and they cannot be directly trained from the data. Model parameters are learned during training when we optimize a loss function using something like gradient descent.

- What degree of polynomial features should I use for my linear model?
- What should be the maximum depth allowed for my decision tree?
- What should be the minimum number of samples required at a leaf node in my decision tree?
- How many trees should I include in my random forest?
- How many neurons should I have in my neural network layer?
- How many layers should I have in my neural network?
- What should I set my learning rate to for gradient descent?

GridSearchCV:

Grid search is a traditional way to perform hyperparameter optimization. It works by searching exhaustively through a specified subset of hyperparameters.Using sklearn's GridSearchCV, we first define our grid of parameters to search over and then run the grid search.



```
from sklearn.model_selection import GridSearchCV
from sklearn.neighbors import KNeighborsClassifier

model = KNeighborsClassifier()
```

```
params = {'n_neighbors': [11,13,17,19,21,23,27,29]}
```

```
model1 = GridSearchCV(model,params,cv=5,verbose=1,n_jobs=10)
```

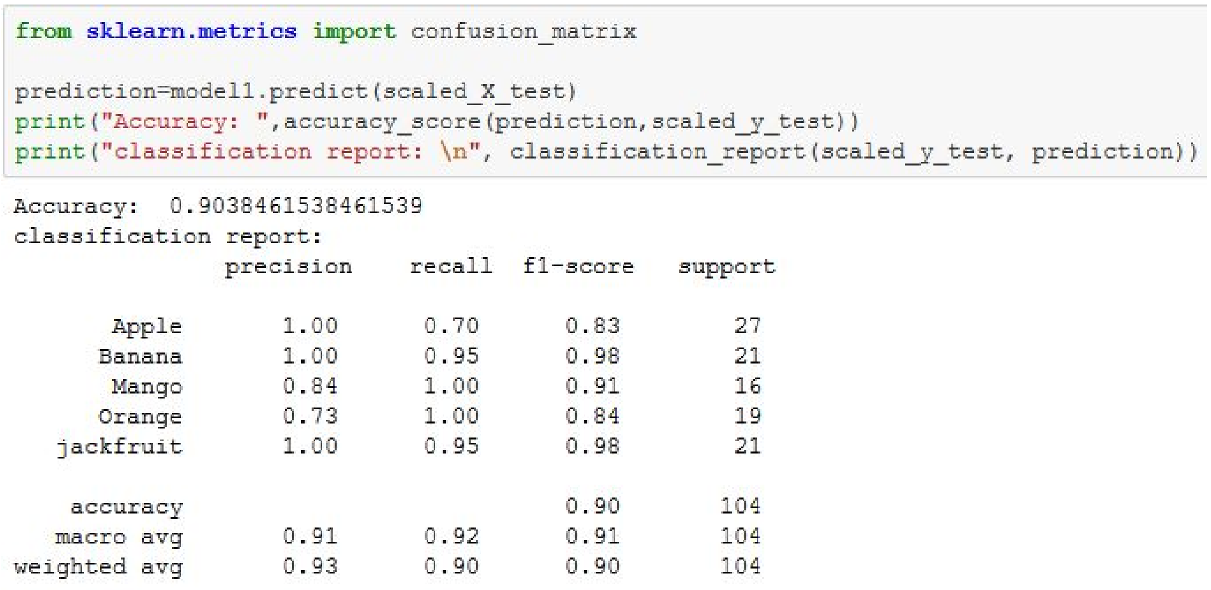
```
model1.fit(scaled_X_train,scaled_y_train)
model1.best_params_
```

```
Fitting 5 folds for each of 8 candidates, totalling 40 fits
[Parallel(n_jobs=10)]: Using backend LokyBackend with 10 concurrent workers.
[Parallel(n_jobs=10)]: Done  40 out of  40 | elapsed:   46.2s finished
{'n_neighbors': 11}
```

Fig 5.3.1.5: KNN with Hyper parameter Tuning

The benefit of grid search is that it is guaranteed to find the optimal combination of parameters supplied. The drawback is that it can be very time consuming and computationally expensive.

Results of HyperParameter tuning:



```
from sklearn.metrics import confusion_matrix

prediction=model1.predict(scaled_X_test)
print("Accuracy: ",accuracy_score(prediction,scaled_y_test))
print("classification report: \n", classification_report(scaled_y_test, prediction))
```

```
Accuracy:  0.9038461538461539
classification report:
               precision    recall  f1-score   support

       Apple       1.00      0.70      0.83        27
      Banana       1.00      0.95      0.98        21
       Mango       0.84      1.00      0.91        16
      Orange       0.73      1.00      0.84        19
   jackfruit       1.00      0.95      0.98        21

    accuracy                           0.90       104
   macro avg       0.91      0.92      0.91       104
weighted avg       0.93      0.90      0.90       104
```

Fig 5.3.1.6: Performance of KNN model

Note: All the results of previous algorithms will be discussed in the Results section.

5.4 VALUE ADDITION:

Adding value to the project i.e we added some more algorithms to the base paper we worked on a bagging technique called RandomForest Classifier and Custom Convolutional neural network.

5.4.1 Random Forest Classifier:

Random forests is a supervised learning algorithm. It can be used both for classification and regression. It is also the most flexible and easy to use the algorithm. A forest consists of trees. It is said that the more trees it has, the more robust a forest is. Random forests create decision trees on randomly selected data samples, get a prediction from each tree, and select the best solution by means of voting. It also provides a pretty good indicator of the feature's importance.

Random forests have a variety of applications, such as recommendation engines, image classification, and feature selection. It can be used to classify loyal loan applicants, identify fraudulent activity, and predict diseases. It lies at the base of the Boruta algorithm, which selects important features in a dataset.

Working of Random Forest:

It works in four steps:

1. Select random samples from a given dataset.
2. Construct a decision tree for each sample and get a prediction result from each decision tree.
3. Perform a vote for each predicted result.
4. Select the prediction result with the most votes as the final prediction.

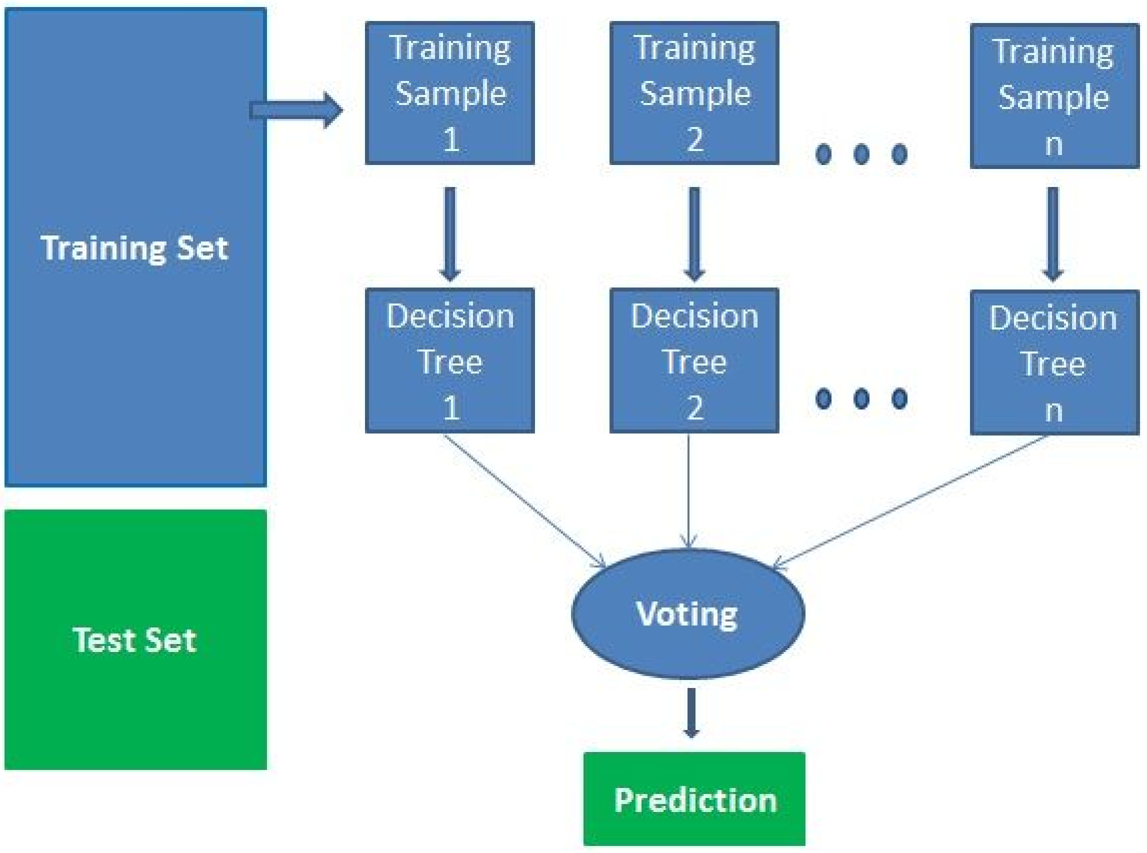


Fig 5.4.1.1: [18] Random Forest Classification

Advantages:

- Random forests is considered as a highly accurate and robust method because of the number of decision trees participating in the process.
- It does not suffer from the overfitting problem. The main reason is that it takes the average of all the predictions, which cancels out the biases.
- The algorithm can be used in both classification and regression problems.
- Random forests can also handle missing values. There are two ways to handle these: using median values to replace continuous variables, and computing the proximity-weighted average of missing values.
- You can get the relative feature importance, which helps in selecting the most contributing features for the classifier.

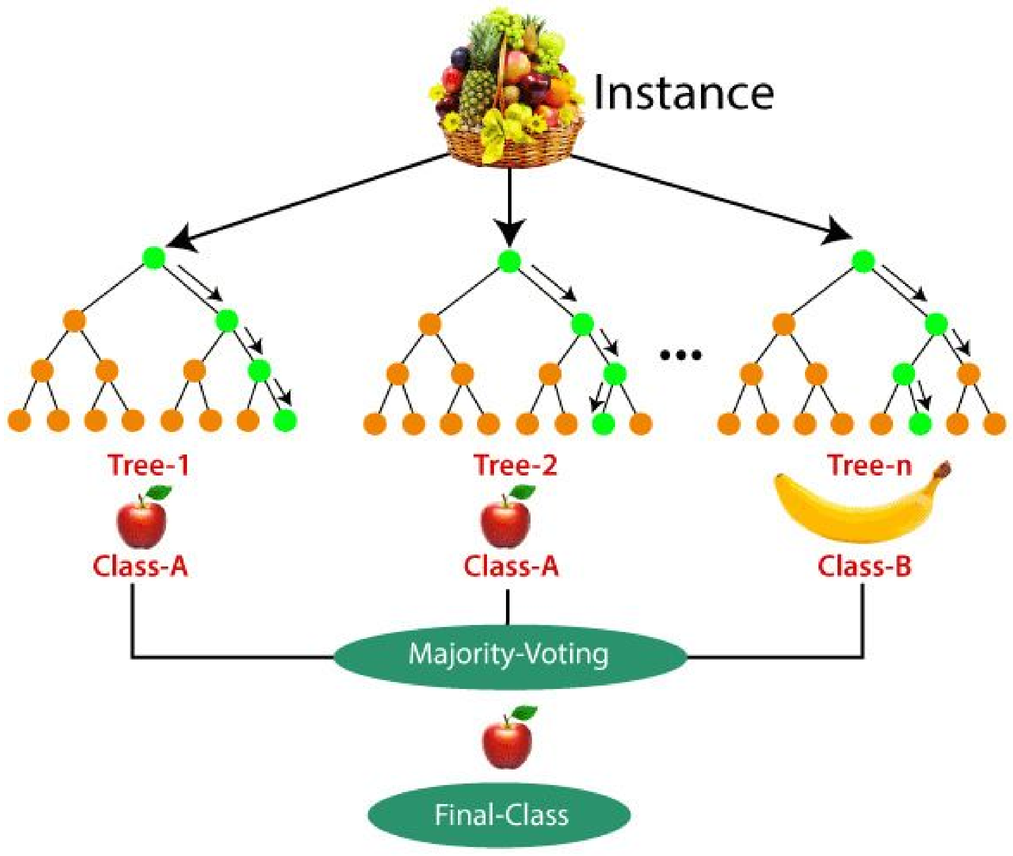
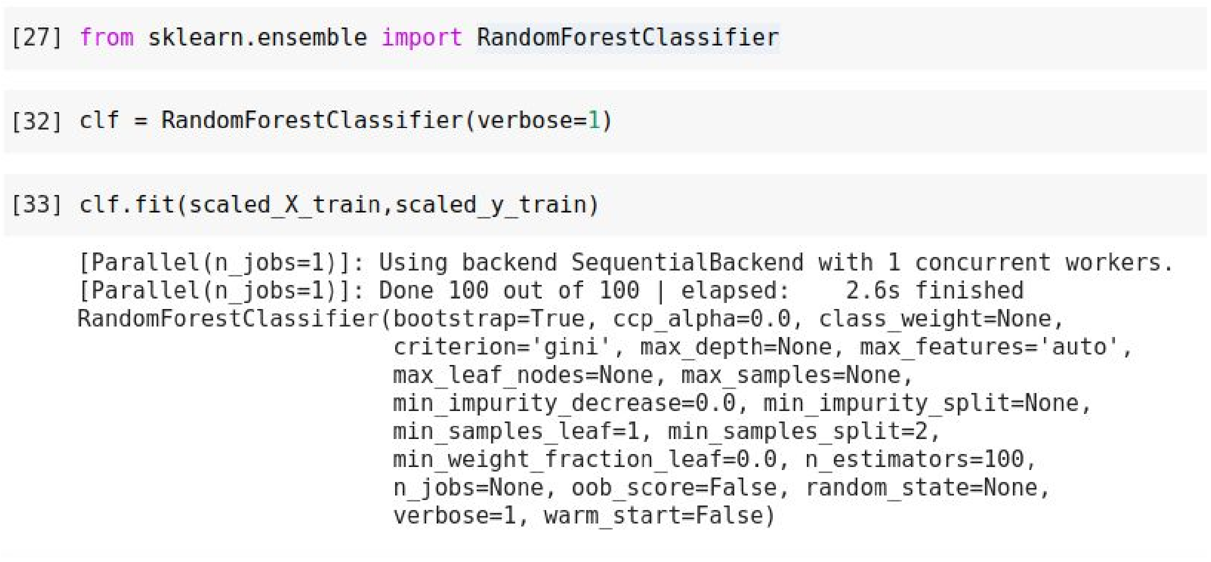


Fig 5.4.1.2: [19] Random Forest Model

Building a Random Forest Classifier:



```
[27] from sklearn.ensemble import RandomForestClassifier

[32] clf = RandomForestClassifier(verbose=1)

[33] clf.fit(scaled_X_train,scaled_y_train)

     [Parallel(n_jobs=1)]: Using backend SequentialBackend with 1 concurrent workers.
     [Parallel(n_jobs=1)]: Done 100 out of 100 | elapsed:    2.6s finished
     RandomForestClassifier(bootstrap=True, ccp_alpha=0.0, class_weight=None,
                            criterion='gini', max_depth=None, max_features='auto',
                            max_leaf_nodes=None, max_samples=None,
                            min_impurity_decrease=0.0, min_impurity_split=None,
                            min_samples_leaf=1, min_samples_split=2,
                            min_weight_fraction_leaf=0.0, n_estimators=100,
                            n_jobs=None, oob_score=False, random_state=None,
                            verbose=1, warm_start=False)
```

Fig 5.4.1.3: Implementation of Random Forest Classifier

Initially, we passed the default parameter, they are:

1. n_estimatorsint, default=100: The number of trees in the forest
2. criterion{"gini", "entropy"}, default="gini": The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain. Note: this parameter is tree-specific.
3. max_depthint, default=None: The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.
4. min_samples_splitint or float, default=2
   The minimum number of samples required to split an internal node.
   a. If int, then consider min_samples_split as the minimum number.
   b. If float, then min_samples_split is a fraction and ceil(min_samples_split * n_samples) are the minimum number of samples for each split.
5. min_samples_leaf*int or float, default=1:* The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.
   a. If int, then consider min_samples_leaf as the minimum number.
   b. If float, then min_samples_leaf is a fraction and ceil(min_samples_leaf * n_samples) are the minimum number of samples for each node.
6. min_weight_fraction_leaffloat, default=0.0: The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.
7. max_features{"auto", "sqrt", "log2"}, int or float, default="auto" The number of features to consider when looking for the best split:
   a. If int, then consider max_features at each split.
   b. If float, then max_features is a fraction and int(max_features * n_features) features are considered at each split.

c. If “auto”, then max_features=sqrt(n_features).
d. If “sqrt”, then max_features=sqrt(n_features) (same as “auto”).
e. If “log2”, then max_features=log2(n_features).
f. If None, then max_features=n_features.

Note: the search for a split does not stop until at least one valid partition of the node samples is found, even if it requires to effectively inspect more than max_features features.

8. max_leaf_nodesint, default=None: Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

9. min_impurity_decreasefloat, default=0.0:A node will be split if this split induces a decrease of the impurity greater than or equal to this value. The weighted impurity decrease equation is the following:

   N_t / N * (impurity - N_t_R / N_t * right_impurity - N_t_L / N_t * left_impurity)

   where N is the total number of samples, N_t is the number of samples at the current node, N_t_L is the number of samples in the left child, and N_t_R is the number of samples in the right child. N, N_t, N_t_R, and N_t_L all refer to the weighted sum if sample_weight is passed.

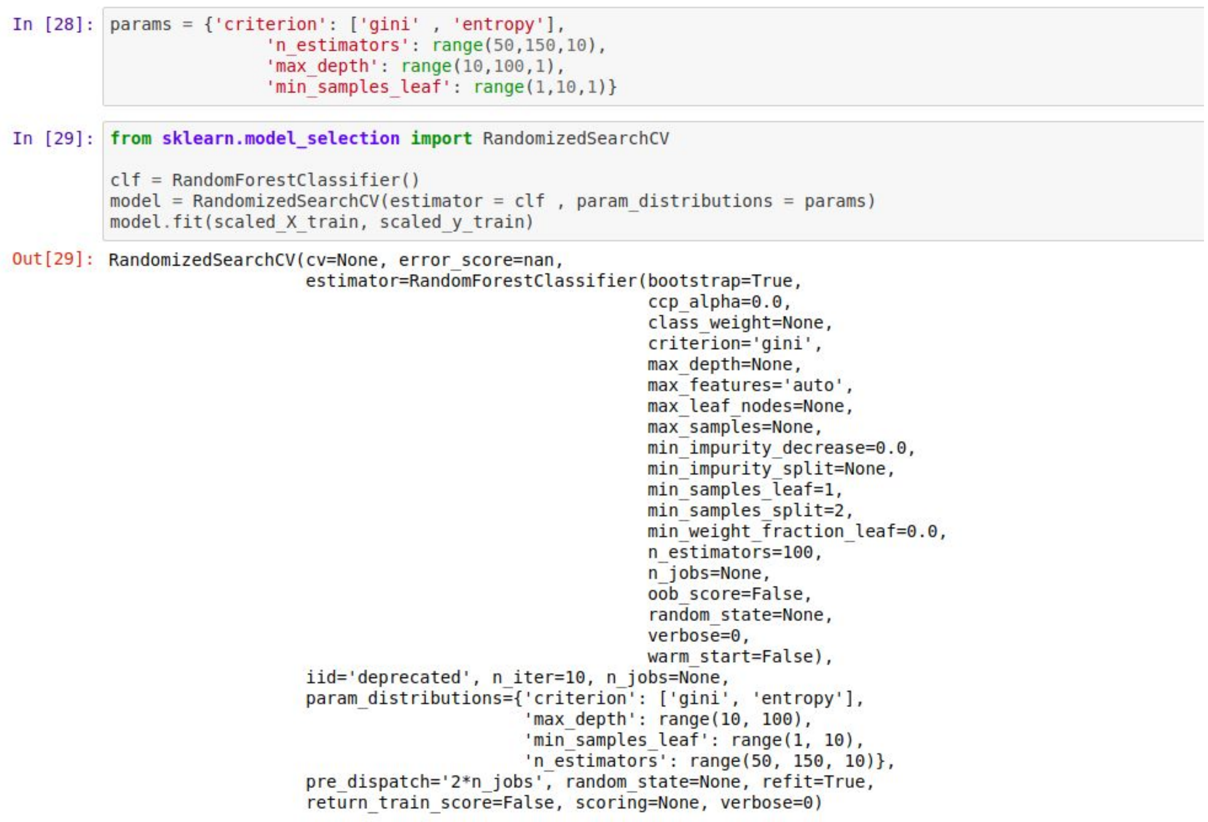
10. min_impurity_splitfloat, default=None: Threshold for early stopping in tree growth. A node will split if its impurity is above the threshold, otherwise, it is a leaf.

11. Bootstrap bool, default=True: Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

Random Forest HyperParameter Tuning:

Hyper parameters in Random Forest : Following are the hyper parameters in Random Forest.

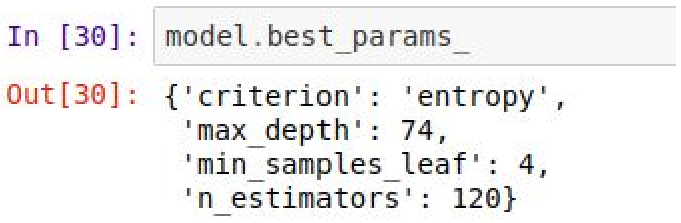
1. N-estimators : Number of decision trees.
2. Criterion : [‘gini’, ‘entropy’]
3. Max_depth : maximum Depth of each Decision Tree.
4. Min samples leaf: min_samples_leaf specifies the minimum number of samples required to be at a leaf node.



```
In [28]: params = {'criterion': ['gini' , 'entropy'],
                   'n_estimators': range(50,150,10),
                   'max_depth': range(10,100,1),
                   'min_samples_leaf': range(1,10,1)}

In [29]: from sklearn.model_selection import RandomizedSearchCV

         clf = RandomForestClassifier()
         model = RandomizedSearchCV(estimator = clf , param_distributions = params)
         model.fit(scaled_X_train, scaled_y_train)

Out[29]: RandomizedSearchCV(cv=None, error_score=nan,
                   estimator=RandomForestClassifier(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    class_weight=None,
                                                    criterion='gini',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None,
                                                    oob_score=False,
                                                    random_state=None,
                                                    verbose=0,
                                                    warm_start=False),
                   iid='deprecated', n_iter=10, n_jobs=None,
                   param_distributions={'criterion': ['gini', 'entropy'],
                                        'max_depth': range(10, 100),
                                        'min_samples_leaf': range(1, 10),
                                        'n_estimators': range(50, 150, 10)},
                   pre_dispatch='2*n_jobs', random_state=None, refit=True,
                   return_train_score=False, scoring=None, verbose=0)
```

Fig 5.4.1.4: Random forest Classifier with Hyperparameter tuning

Best parameters are:



```
In [30]: model.best_params_

Out[30]: {'criterion': 'entropy',
          'max_depth': 74,
          'min_samples_leaf': 4,
          'n_estimators': 120}
```

Fig 5.4.1.5: Best parameters for Random Forest Classifier

5.4.2 Convolutional Neural Network:

A Convolutional Neural Network (ConvNet/CNN) is a Deep Learning algorithm which can take in an input image, assign importance (learnable weights and biases) to various aspects/objects in the image and be able to differentiate one from the other. The pre-processing required in a ConvNet is much lower as compared to other classification algorithms. While in primitive methods filters are hand-engineered, with enough training, ConvNets have the ability to learn these filters/characteristics.

The architecture of a ConvNet is analogous to that of the connectivity pattern of Neurons in the Human Brain and was inspired by the organization of the Visual Cortex. Individual neurons respond to stimuli only in a restricted region of the visual field known as the Receptive Field. A collection of such fields overlap to cover the entire visual area.

A ConvNet is able to successfully capture the Spatial and Temporal dependencies in an image through the application of relevant filters. The architecture performs a better fitting to the image dataset due to the reduction in the number of parameters involved and the reusability of weights. In other words, the network can be trained to understand the sophistication of the image better.

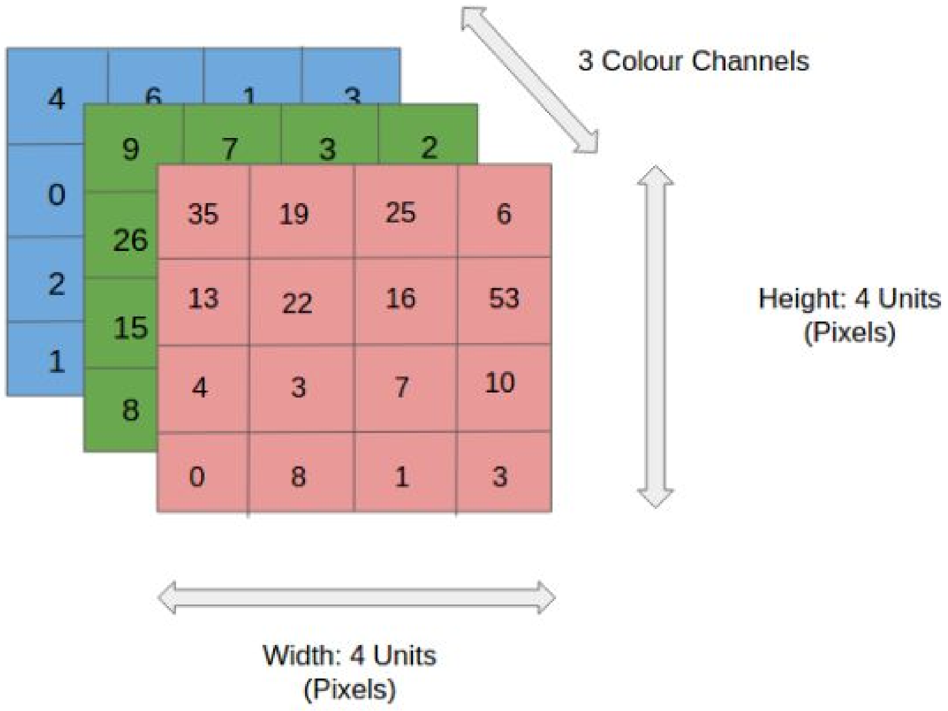


Fig 5.4.2.1: [20] RGB image (CNN)

In the figure, we have an RGB image which has been separated by its three color planes — Red, Green, and Blue. There are a number of such color spaces in which images exist — Grayscale, RGB, HSV, CMYK, etc.

We can imagine how computationally intensive things would get once the images reach dimensions, say 8K (7680×4320). The role of the ConvNet is to reduce the images into a form that is easier to process, without losing features which are critical for getting a good prediction. This is important when we are to design an architecture that is not only good at learning features but also is scalable to massive datasets.

The Convolutional Neural network works in the form of different kinds of layers.

Convolution Layer:

Convolution is the first layer to extract features from an input image. Convolution preserves the relationship between pixels by learning image features using small squares of input data. It is a mathematical operation that takes two inputs such as an image matrix and a filter or kernel.

- An image matrix (volume) of dimension **(h x w x d)**
- A filter **(f_h x f_w x d)**
- Outputs a volume dimension **(h - f_h + 1) x (w - f_w + 1) x 1**

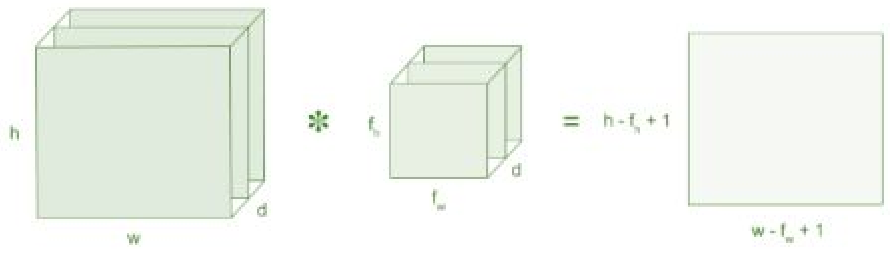
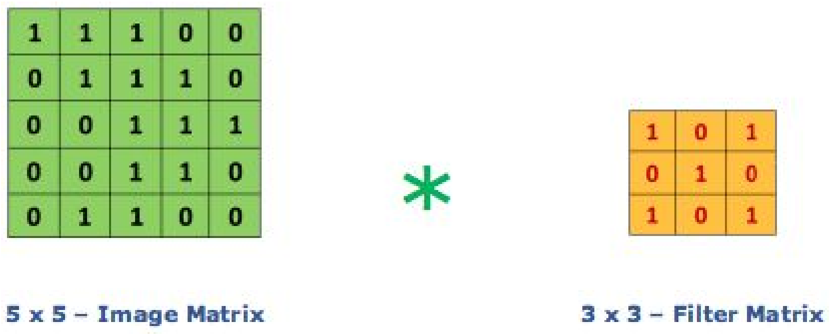


Fig 5.4.2.2: [20] Convolution Layer

Consider a 5 x 5 whose image pixel values are 0, 1 and filter matrix 3 x 3 as shown in below.



1	1	1	0	0
0	1	1	1	0
0	0	1	1	1
0	0	1	1	0
0	1	1	0	0

*

1	0	1
0	1	0
1	0	1

5 x 5 – Image Matrix
3 x 3 – Filter Matrix

Fig 5.4.2.3: [20] Feature Map

Then the convolution of the 5 x 5 image matrix multiplied with 3 x 3 filter matrix which is called **“Feature Map”** as output shown in below.

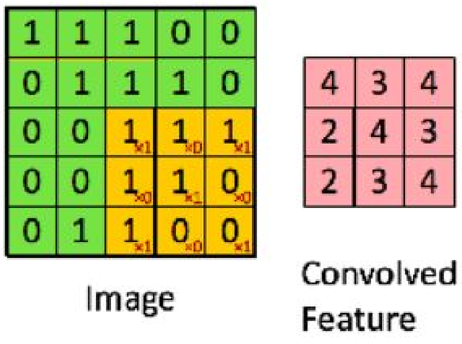


Fig 5.4.2.4: [20] Convolved Feature

Strides:

Stride is the number of pixels shifted over the input matrix. When the stride is 1 then we move the filters to 1 pixel at a time. When the stride is 2 then we move the filters to 2 pixels at a time and so on. The below figure shows convolution would work with a stride of 2.

Padding

Sometimes filters do not perfectly fit the input image. We have two options:

- Pad the picture with zeros (zero-padding) so that it fits.
- Drop the part of the image where the filter did not fit. This is called valid padding which keeps only valid parts of the image.

Non Linearity (ReLU)

ReLU stands for Rectified Linear Unit for a non-linear operation. The output is ***f(x) = max(0,x).***

Why ReLU is important: ReLU's purpose is to introduce non-linearity in our ConvNet. Since the real-world data would want our ConvNet to learn would be non-negative linear values.

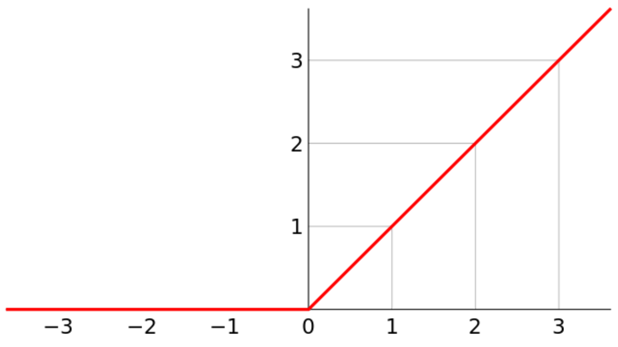


Fig 5.4.2.5: [21] Non Linearity (ReLU)

Pooling Layer

Pooling layers section would reduce the number of parameters when the images are too large. Spatial pooling is also called subsampling or downsampling which reduces the dimensionality of each map but retains important information. Spatial Pooling can be of different types:

- Max Pooling
- Average Pooling
- Sum Pooling

Max pooling takes the largest element from the rectified feature map. Taking the largest element could also take the average pooling. Sum of all elements in the feature map call as sum pooling.

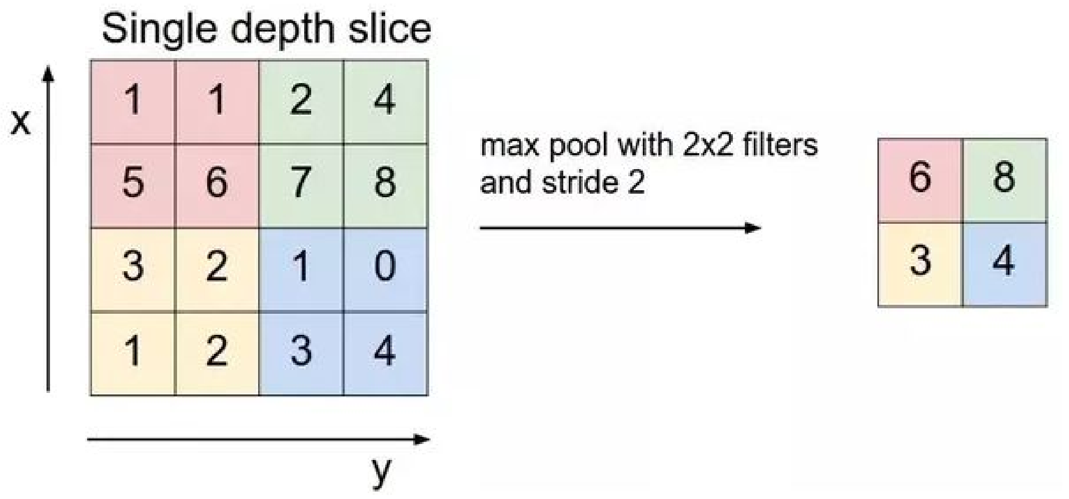


Fig 5.4.2.6: [20] Max pooling

Fully Connected Layer

The layer we call the FC layer, we flatten our matrix into a vector and feed it into a fully connected layer like a neural network.

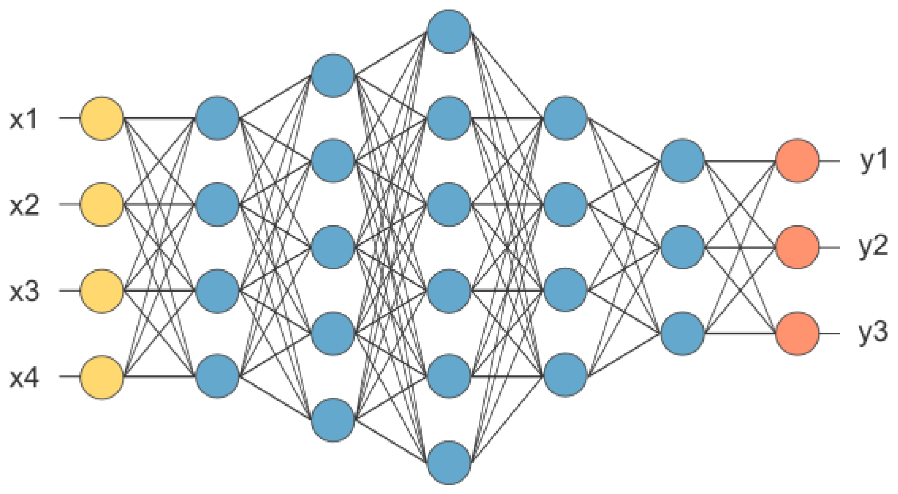
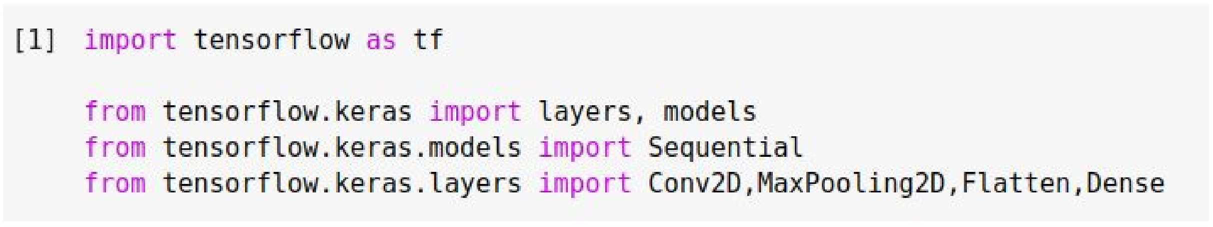


Fig 5.4.2.7: Fully Connected Layer

Implementation of Convolutional neural network:

To implement the convolutional neural network we are using an open source Module named TensorFlow version 2.2

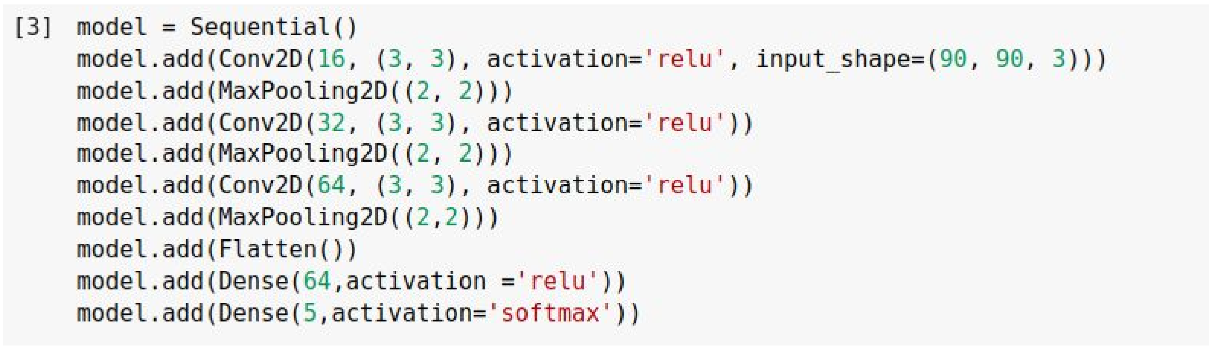
Firstly we need to import the necessary modules and classes.



```
[1]  import tensorflow as tf

     from tensorflow.keras import layers, models
     from tensorflow.keras.models import Sequential
     from tensorflow.keras.layers import Conv2D,MaxPooling2D,Flatten,Dense
```

Fig 5.4.2.8: Importing packages (CNN)

Now we need to create a Convolutional neural network i.e we need to build our architecture.



```
[3]  model = Sequential()
     model.add(Conv2D(16, (3, 3), activation='relu', input_shape=(90, 90, 3)))
     model.add(MaxPooling2D((2, 2)))
     model.add(Conv2D(32, (3, 3), activation='relu'))
     model.add(MaxPooling2D((2, 2)))
     model.add(Conv2D(64, (3, 3), activation='relu'))
     model.add(MaxPooling2D((2,2)))
     model.add(Flatten())
     model.add(Dense(64,activation ='relu'))
     model.add(Dense(5,activation='softmax'))
```

Fig 5.4.2.9: Building Convolution Neural Network

The architecture of custom CNN:

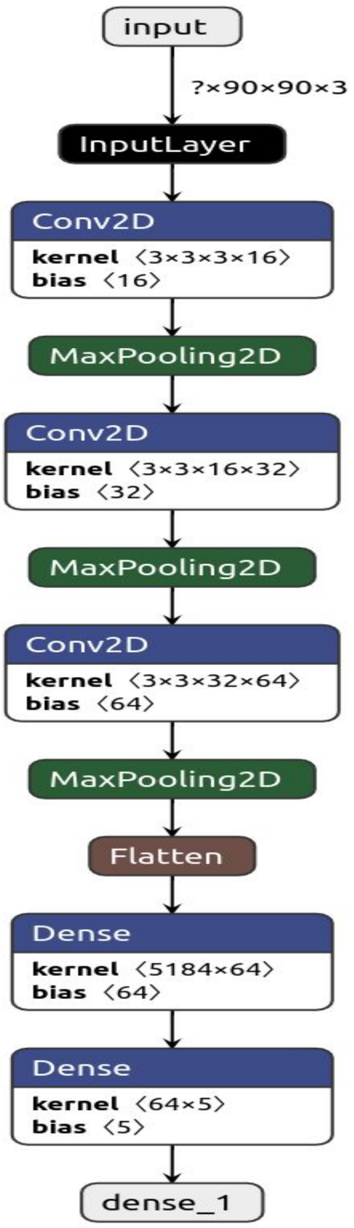
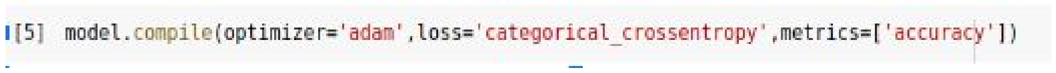


Fig 5.4.2.10: Architecture of CNN

1. The first layer is a convolutional layer which is also an input layer.
2. The next layer is a Maxpooling2D layer which collects the important features of the image.
3. This layer is followed by a Convolutional layer and the input is a convoluted image from the first Convolution layer.
4. This convolutional layer is followed by a Maxpooling2D layer.

5. Again this layer is followed by a Convolutional layer and input to this layer is a convoluted image from the previous maxpooling2d layer.
6. Finally, there is a max-pooling layer that ends the 2d kernel learning.
7. Now there is a Flatten layer that converts 2d coordinates to 1d array.
8. After the Flatten layer is added now we need to add a series of dense layers to identify the exact class of the prediction and change the weights accordingly.
9. At last a dense layer is added with softmax as activation to classify the images

Now we need to compile the model with optimizer and loss function.



```
[5] model.compile(optimizer='adam',loss='categorical_crossentropy',metrics=['accuracy'])
```

Fig 5.4.2.11: Compiling CNN model

Adam is an optimization algorithm that can be used instead of the classical stochastic gradient descent procedure to update network weights iteratively based on training data.

Adam was presented by Diederik Kingma from OpenAI and Jimmy Ba from the University of Toronto in their 2015 ICLR paper (poster) titled “Adam: A Method for Stochastic Optimization“. I will quote liberally from their paper in this post, unless stated otherwise.

Specifically, the algorithm calculates an exponential moving average of the gradient and the squared gradient, and the parameters beta1 and beta2 control the decay rates of these moving averages.

The initial value of the moving averages and beta1 and beta2 values close to 1.0 (recommended) result in a bias of moment estimates towards zero. This bias is overcome by first calculating the biased estimates before then calculating bias-corrected estimates.

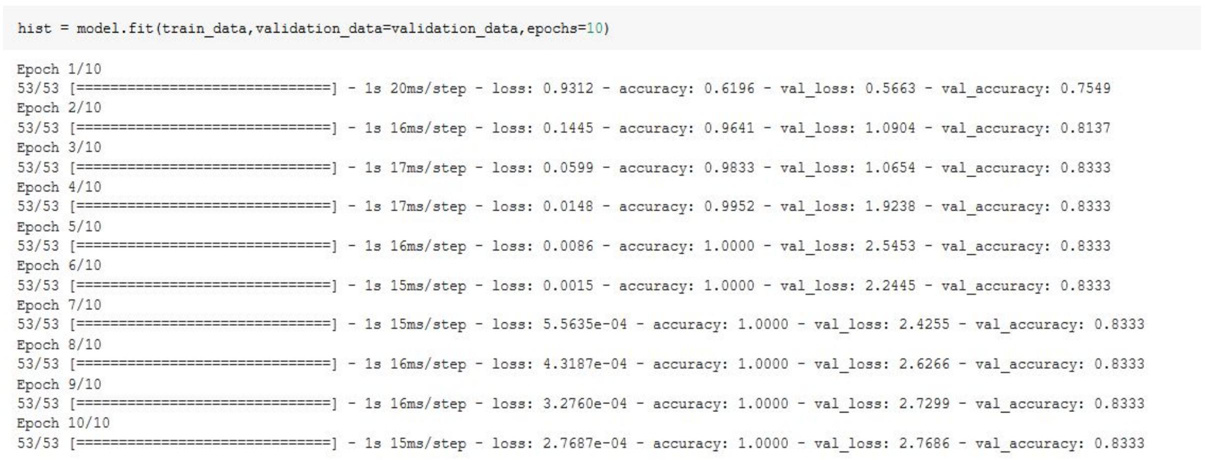
Adam parameters:

- alpha. Also referred to as the learning rate or step size. The proportion that weights are updated (e.g. 0.001). Larger values (e.g. 0.3) results in faster initial learning before the rate is updated. Smaller values (e.g. 1.0E-5) slow learning right down during training
- beta1. The exponential decay rate for the first moment estimates (e.g. 0.9).

- beta2. The exponential decay rate for the second-moment estimates (e.g. 0.999). This value should be set close to 1.0 on problems with a sparse gradient (e.g. NLP and computer vision problems).
- epsilon. Is a very small number to prevent any division by zero in the implementation (e.g. 10E-8).

Loss function: Categorical cross-entropy: a loss function is a function that calculates the value difference between the actual value and the predicted value. Categorical cross-entropy is a loss function that is used in multi-class classification tasks.

After compiling the model we need to fit the model and train it over our data.



```
hist = model.fit(train_data,validation_data=validation_data,epochs=10)
```

```
Epoch 1/10
53/53 [==============================] - 1s 20ms/step - loss: 0.9312 - accuracy: 0.6196 - val_loss: 0.5663 - val_accuracy: 0.7549
Epoch 2/10
53/53 [==============================] - 1s 16ms/step - loss: 0.1445 - accuracy: 0.9641 - val_loss: 1.0904 - val_accuracy: 0.8137
Epoch 3/10
53/53 [==============================] - 1s 17ms/step - loss: 0.0599 - accuracy: 0.9833 - val_loss: 1.0654 - val_accuracy: 0.8333
Epoch 4/10
53/53 [==============================] - 1s 17ms/step - loss: 0.0148 - accuracy: 0.9952 - val_loss: 1.9238 - val_accuracy: 0.8333
Epoch 5/10
53/53 [==============================] - 1s 16ms/step - loss: 0.0086 - accuracy: 1.0000 - val_loss: 2.5453 - val_accuracy: 0.8333
Epoch 6/10
53/53 [==============================] - 1s 15ms/step - loss: 0.0015 - accuracy: 1.0000 - val_loss: 2.2445 - val_accuracy: 0.8333
Epoch 7/10
53/53 [==============================] - 1s 15ms/step - loss: 5.5635e-04 - accuracy: 1.0000 - val_loss: 2.4255 - val_accuracy: 0.8333
Epoch 8/10
53/53 [==============================] - 1s 16ms/step - loss: 4.3187e-04 - accuracy: 1.0000 - val_loss: 2.6266 - val_accuracy: 0.8333
Epoch 9/10
53/53 [==============================] - 1s 16ms/step - loss: 3.2760e-04 - accuracy: 1.0000 - val_loss: 2.7299 - val_accuracy: 0.8333
Epoch 10/10
53/53 [==============================] - 1s 15ms/step - loss: 2.7687e-04 - accuracy: 1.0000 - val_loss: 2.7686 - val_accuracy: 0.8333
```

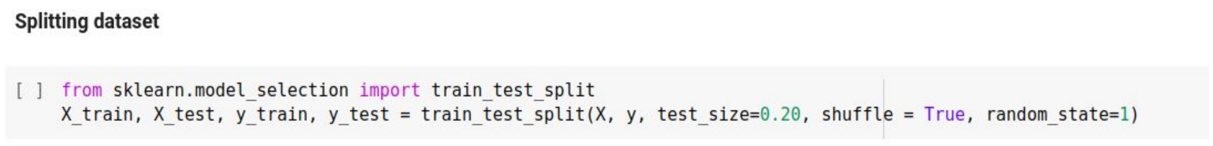
Fig 5.4.2.12: CNN model training and validation

6. TESTING AND VALIDATION

Multi-Class:

Classification task with more than two classes such that the input is to be classified into one, and only one of these classes. Example: classify a set of images of fruits into any one of these categories — apples, bananas, and oranges.
Testing and validating a machine learning or deep learning model has following steps:

Initially we needed to collect the test data, but in this case we had split the data into training and testing sets from a collection of data as shown in the fig 32.



Splitting dataset

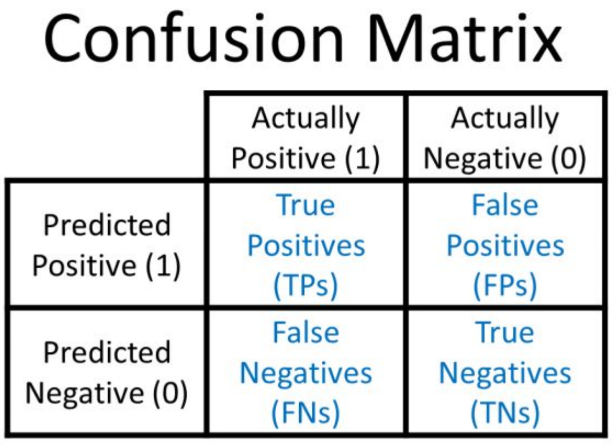
```
[ ]  from sklearn.model_selection import train_test_split
     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, shuffle = True, random_state=1)
```

Fig 6.1: Splitting data for testing model

Now all the test cases that will be considered will be there in the test set, i.e. in X_test we will be having the image data and in y_test , labels or the corresponding X_test will be present.

There are certain metrics to evaluate a machine learning model, following are the metrics presented in this project to evaluate our models.

- Accuracy: It's the ratio of the correctly labeled subjects to the whole pool of subjects.
- Precision: it is the ratio of the correctly positively labeled by our program to all positively labeled.
- Sensitivity: It is the ratio of correctly classified objects which are really correct
- Specificity: It is the ratio of correctly classified negative objects as negative.
- F1 Score: it is the harmonic mean of Precision and Sensitivity
- Confusion Matrix: A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. And elements of the confusion matrix are below.



Confusion Matrix

	Actually Positive (1)	Actually Negative (0)
Predicted Positive (1)	True Positives (TPs)	False Positives (FPs)
Predicted Negative (0)	False Negatives (FNs)	True Negatives (TNs)

Fig 6.2: Confusion matrix

  - True positives (TP): These are cases in which we predicted yes (they have the disease), and they do have the disease.
  - True negatives (TN): We predicted no, and they don't have the disease.
  - False positives (FP): We predicted yes, but they don't actually have the disease. (Also known as a "Type I error.")
  - False negatives (FN): We predicted no, but they actually do have the disease. (Also known as a "Type II error.")
- **Accuracy:** Overall, how often is the classifier correct?
  - (TP+TN)/total
- **Misclassification Rate:** Overall, how often is it wrong?
  - (FP+FN)/total
  - equivalent to 1 minus Accuracy also known as "Error Rate"
- **True Positive Rate:** When it's actually yes, how often does it predict yes?
  - TP/actual
  - also known as "Sensitivity" or "Recall"
- **False Positive Rate:** When it's actually no, how often does it predict yes?
  - FP/actual no
- **True Negative Rate:** When it's actually no, how often does it predict no?
  - TN/actual no
  - equivalent to 1 minus False Positive Rate
  - also known as "Specificity"

- **Precision:** When it predicts yes, how often is it correct?
    - TP/predicted
- **Prevalence:** How often does the yes condition actually occur in our sample?
    - actual yes/total

Cross-validation:

Cross-validation is a resampling procedure used to evaluate machine learning models on a limited data sample. The procedure has a single parameter called k that refers to the number of groups that a given data sample is to be split into. As such, the procedure is often called k-fold cross-validation.

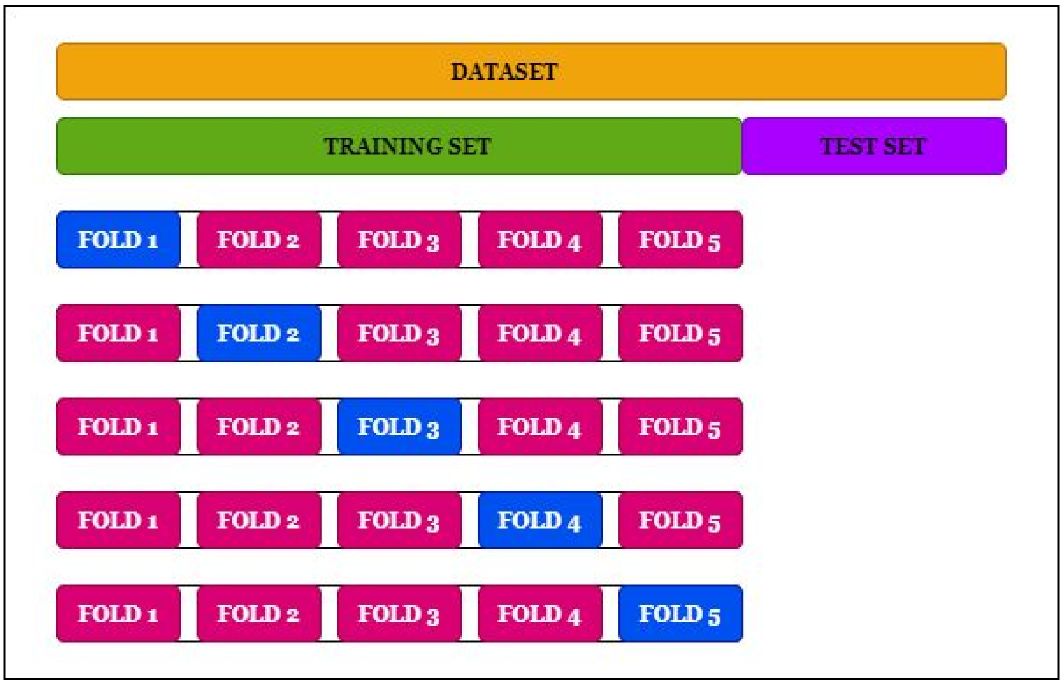


Fig 6.3: Cross Validation

Cross-validation is primarily used in applied machine learning to estimate the skill of a machine learning model on unseen data. That is, to use a limited sample in order to estimate how the model is expected to perform in general when used to make predictions on data not used during the training of the model.

It is a popular method because it is simple to understand and because it generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split.

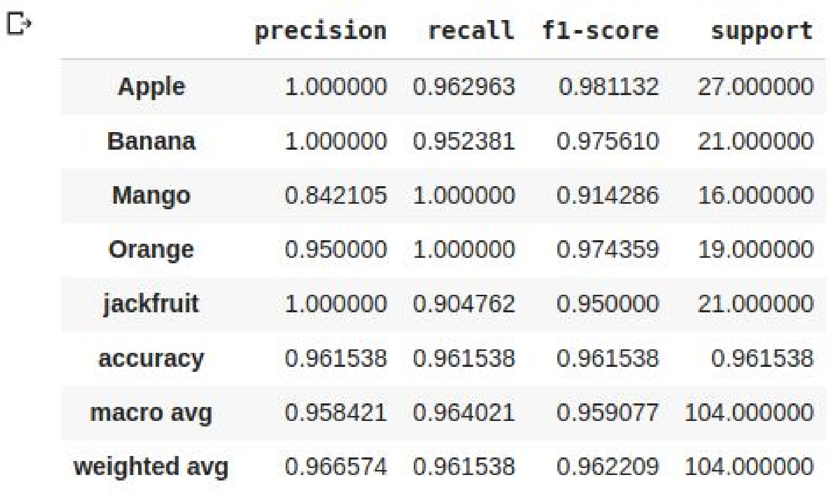
The general procedure is as follows:

1. Shuffle the dataset randomly.
2. Split the dataset into k groups
3. For each unique group:
    1. Take the group as a holdout or test data set
    2. Take the remaining groups as a training data set
    3. Fit a model on the training set and evaluate it on the test set
    4. Retain the evaluation score and discard the model
4. Summarize the skill of the model using the sample of model evaluation scores

Importantly, each observation in the data sample is assigned to an individual group and stays in that group for the duration of the procedure. This means that each sample is given the opportunity to be used in the hold-out set 1 time and used to train the model k-1 times.

7. RESULT ANALYSIS

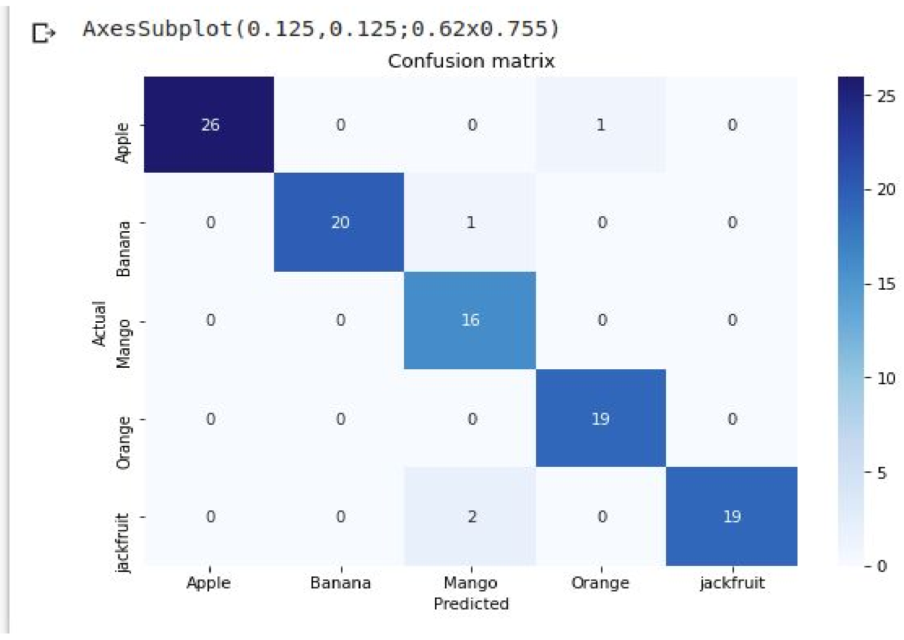
KNN unscaled data results over the test set:



	precision	recall	f1-score	support
Apple	1.000000	0.962963	0.981132	27.000000
Banana	1.000000	0.952381	0.975610	21.000000
Mango	0.842105	1.000000	0.914286	16.000000
Orange	0.950000	1.000000	0.974359	19.000000
jackfruit	1.000000	0.904762	0.950000	21.000000
accuracy	0.961538	0.961538	0.961538	0.961538
macro avg	0.958421	0.964021	0.959077	104.000000
weighted avg	0.966574	0.961538	0.962209	104.000000

Fig 7.1: KNN results over unscaled data

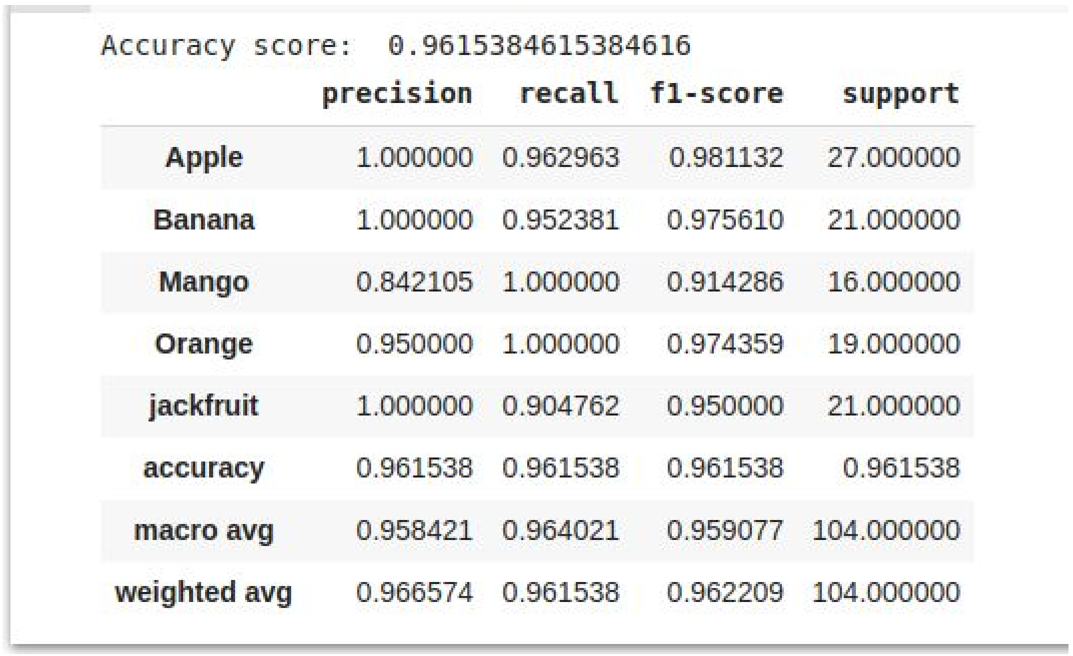
Confusion Matrix:



```
AxesSubplot(0.125,0.125;0.62x0.755)
```

Fig 7.2: Confusion matrix of KNN (unscaled data)

KNN results over-scaled data:

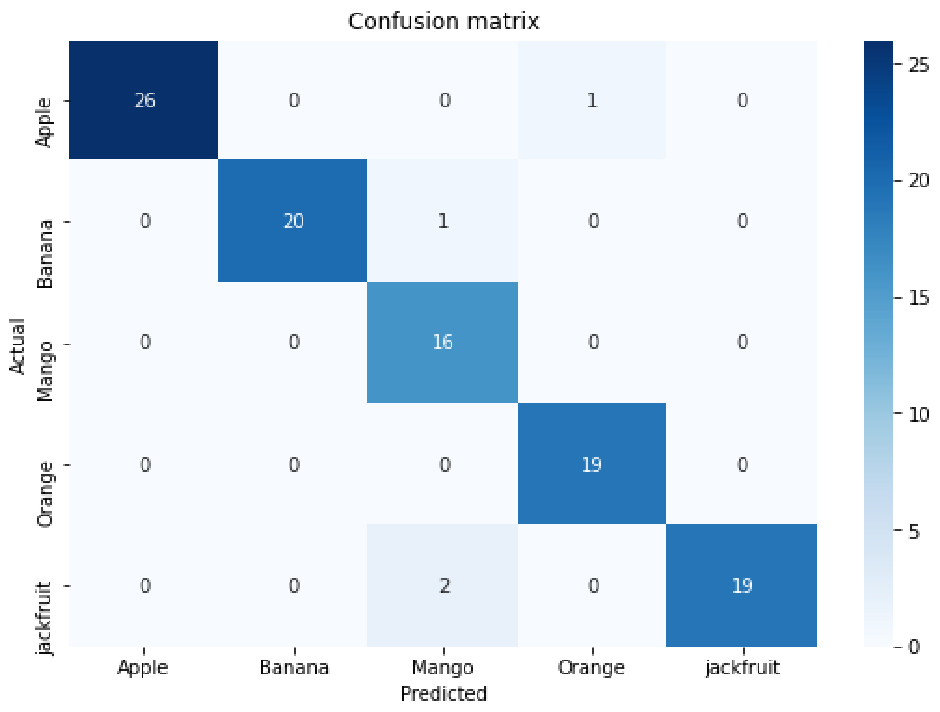


```
Accuracy score:  0.9615384615384616
```

	precision	recall	f1-score	support
Apple	1.000000	0.962963	0.981132	27.000000
Banana	1.000000	0.952381	0.975610	21.000000
Mango	0.842105	1.000000	0.914286	16.000000
Orange	0.950000	1.000000	0.974359	19.000000
jackfruit	1.000000	0.904762	0.950000	21.000000
accuracy	0.961538	0.961538	0.961538	0.961538
macro avg	0.958421	0.964021	0.959077	104.000000
weighted avg	0.966574	0.961538	0.962209	104.000000

Fig 7.3: KNN results over - scaled Data

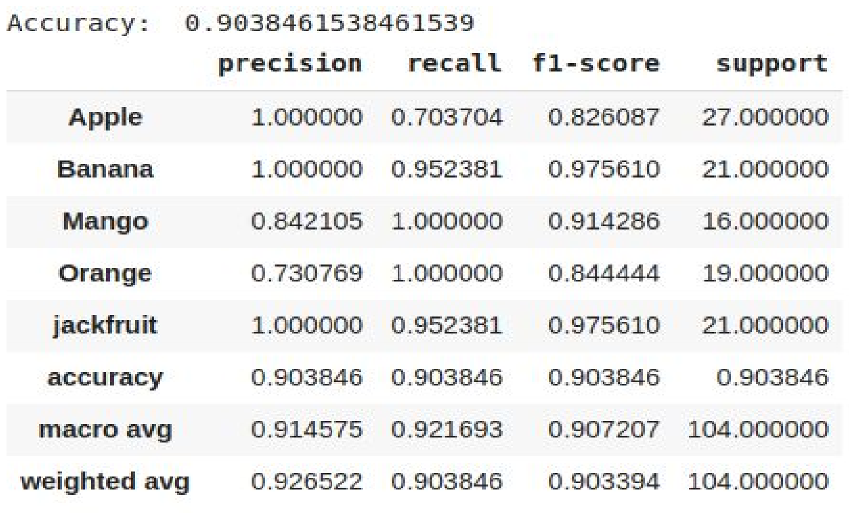
Confusion Matrix:



Actual \ Predicted	Apple	Banana	Mango	Orange	jackfruit
Apple	26	0	0	1	0
Banana	0	20	1	0	0
Mango	0	0	16	0	0
Orange	0	0	0	19	0
jackfruit	0	0	2	0	19

Fig 7.4: Confusion matrix of KNN over - scaled data

Results of Hyperparameter tuned KNN:

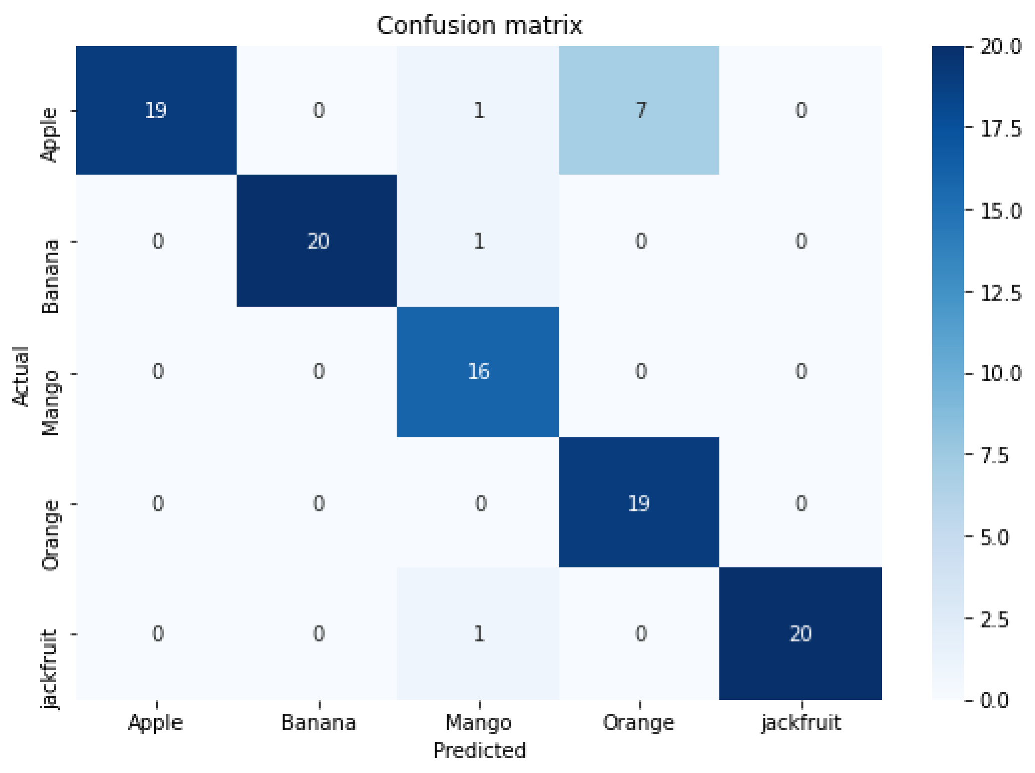


```
Accuracy:  0.9038461538461539
```

	precision	recall	f1-score	support
Apple	1.000000	0.703704	0.826087	27.000000
Banana	1.000000	0.952381	0.975610	21.000000
Mango	0.842105	1.000000	0.914286	16.000000
Orange	0.730769	1.000000	0.844444	19.000000
jackfruit	1.000000	0.952381	0.975610	21.000000
accuracy	0.903846	0.903846	0.903846	0.903846
macro avg	0.914575	0.921693	0.907207	104.000000
weighted avg	0.926522	0.903846	0.903394	104.000000

Fig 7.5: Results of Hyperparameter tuned KNN

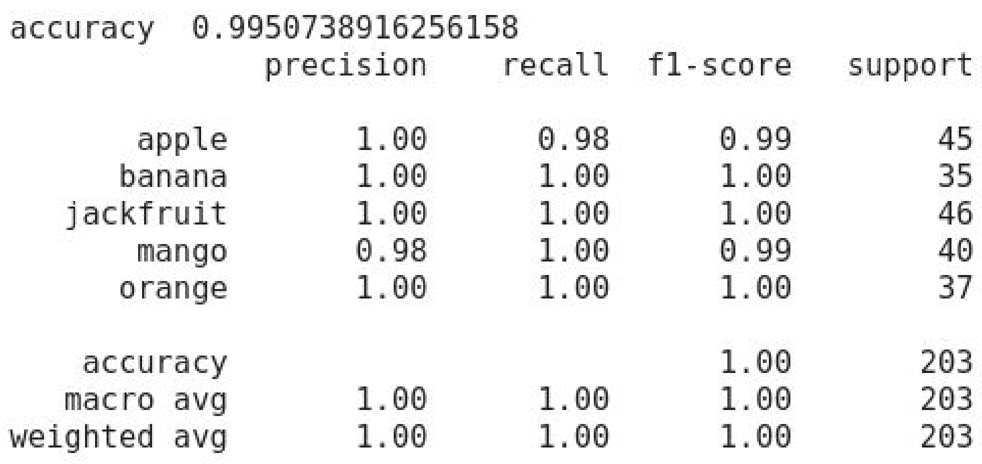
Confusion Matrix:



Actual \ Predicted	Apple	Banana	Mango	Orange	jackfruit
Apple	19	0	1	7	0
Banana	0	20	1	0	0
Mango	0	0	16	0	0
Orange	0	0	0	19	0
jackfruit	0	0	1	0	20

Fig 7.6: Confusion matrix of Hyperparameter tuned KNN

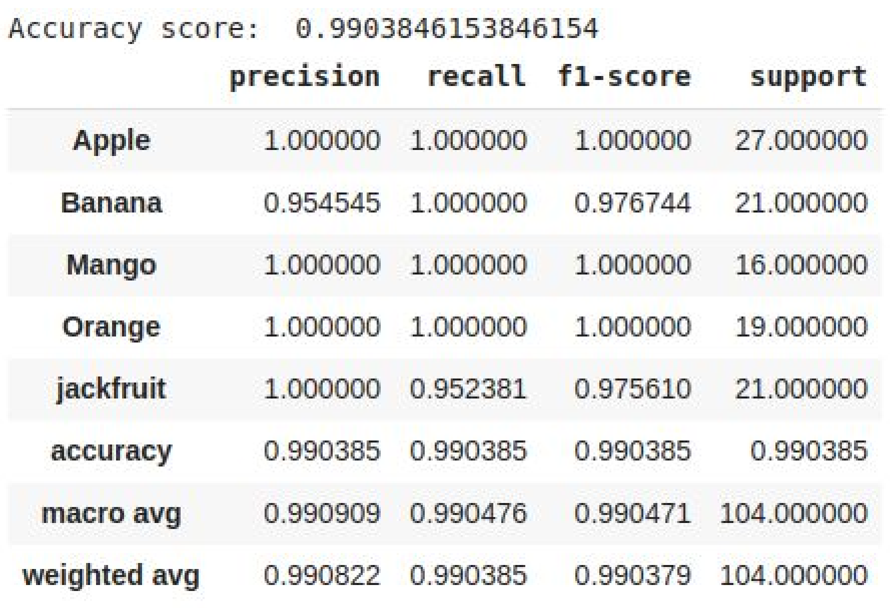
Random Forest Results:



```
accuracy  0.9950738916256158
              precision    recall  f1-score   support

       apple       1.00      0.98      0.99        45
      banana       1.00      1.00      1.00        35
   jackfruit       1.00      1.00      1.00        46
       mango       0.98      1.00      0.99        40
      orange       1.00      1.00      1.00        37

    accuracy                           1.00       203
   macro avg       1.00      1.00      1.00       203
weighted avg       1.00      1.00      1.00       203
```

Fig 7.7: Results of Random Forest Classifier

Results of Random Forest Hyperparameter tuned:



Accuracy score: 0.9903846153846154

	precision	recall	f1-score	support
Apple	1.000000	1.000000	1.000000	27.000000
Banana	0.954545	1.000000	0.976744	21.000000
Mango	1.000000	1.000000	1.000000	16.000000
Orange	1.000000	1.000000	1.000000	19.000000
jackfruit	1.000000	0.952381	0.975610	21.000000
accuracy	0.990385	0.990385	0.990385	0.990385
macro avg	0.990909	0.990476	0.990471	104.000000
weighted avg	0.990822	0.990385	0.990379	104.000000

Fig 7.8: Confusion matrix of Hyperparameter tuned Random Forest Classifier

Confusion Matrix of Random Forest Hyperparameter tuned:

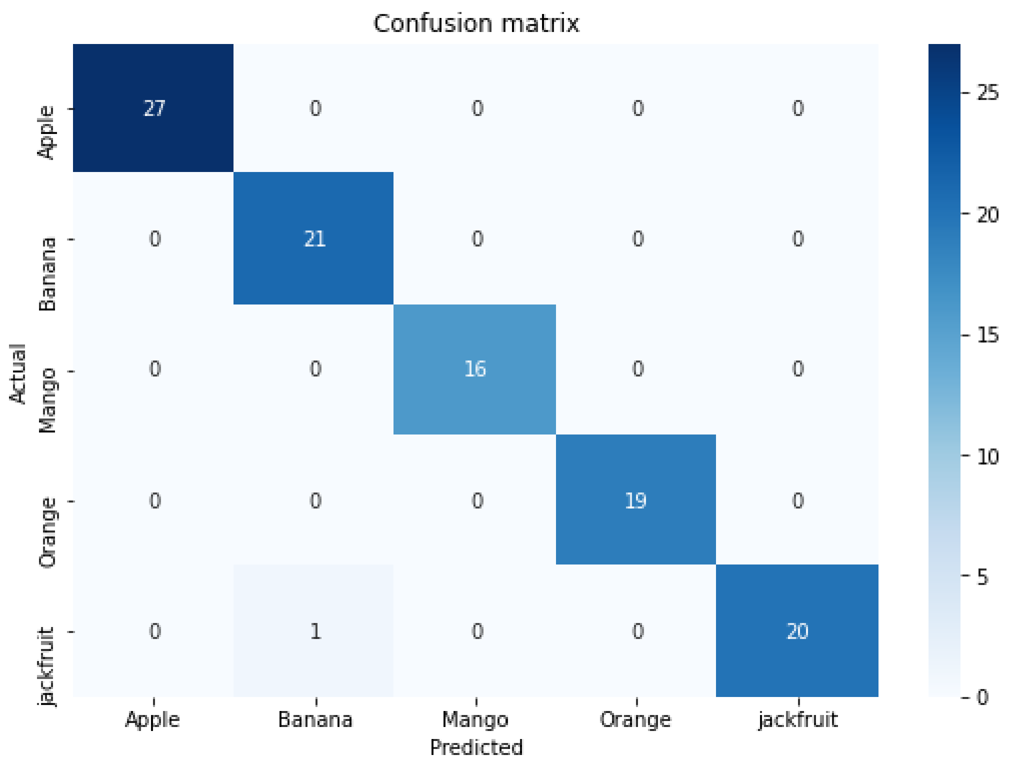
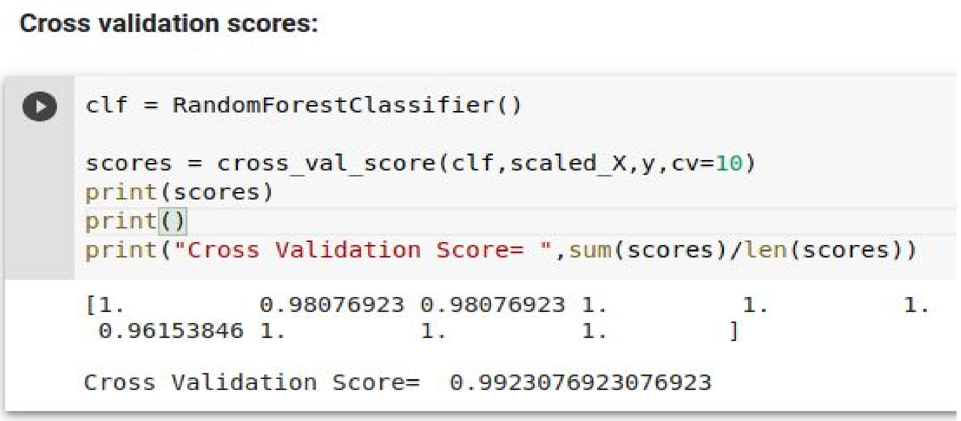


Fig 7.9: Confusion Matrix of Random Forest Hyperparameter tuned

As the accuracy of the random forest was very high there was a probability of the model to Overfit over the data to address this issue we have a method called Cross-Validation Score.



Cross validation scores:

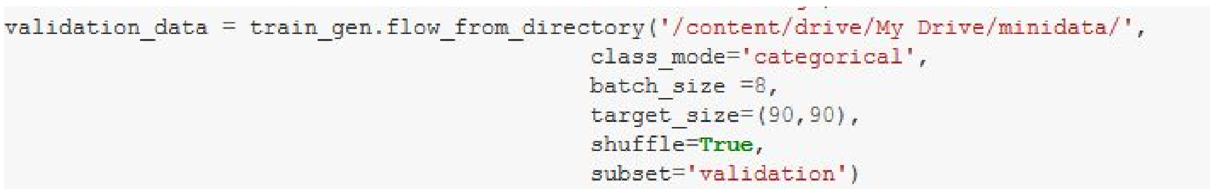
```
clf = RandomForestClassifier()

scores = cross_val_score(clf,scaled_X,y,cv=10)
print(scores)
print()
print("Cross Validation Score= ",sum(scores)/len(scores))
```

```
[1.         0.98076923 0.98076923 1.         1.         1.
 0.96153846 1.         1.         1.        ]

Cross Validation Score=  0.9923076923076923
```

Fig 7.10: Cross-validation scores of Random Forest Classifier

CNN Result Analysis:

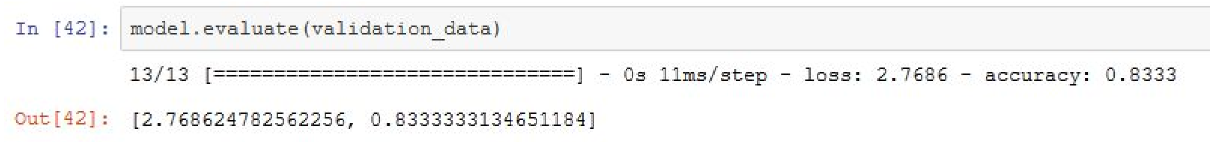
After training a multi class classification problem using Convolutional Neural Network (CNN), we initially need to load our validation data in the required format i.e., the rescaled image data, to do that we used a module named ImageDataGenerator with parameters given to rescale the images and in the below fig: 45 we have taken 20% of our whole dataset and made it to be completely unseen data i.e., we did not use this portion of the dataset for training our custom CNN.



```
validation_data = train_gen.flow_from_directory('/content/drive/My Drive/minidata/',
                                                class_mode='categorical',
                                                batch_size =8,
                                                target_size=(90,90),
                                                shuffle=True,
                                                subset='validation')
```

Fig 7.11: Validation data

As we used an open source library developed and maintained by Google named “Tensorflow”, this module helps the developer by making the evaluation process very much simpler and very much easy to evaluate the training model.



```
In [42]: model.evaluate(validation_data)

         13/13 [==============================] - 0s 11ms/step - loss: 2.7686 - accuracy: 0.8333

Out[42]: [2.768624782562256, 0.8333333134651184]
```

Fig 7.12: Results of CNN

In fig 46, we observed that over the test data is 83% and the loss value over the data is 2.77.

8. CONCLUSION

In the whole documentation of the project to identify different kinds of fruits, we analyzed the base paper and analyzed new algorithms namely Random forest and Convolutional neural Network. All the phases of machine learning like data collection, data preprocessing, model selection, training and results in the analysis are detailedly analyzed in the above documentation.

As a future aspect of the project, we can use transfer learning-based pre-trained convolutional neural Networks, like VGG19, Resnet, InceptionV2, furthermore, we can make the project available in real-time using object Detection frameworks like SSD and RCNN.

REFERENCES

[1] ABARE. Australian Vegetable Growing Farms: An Economic Survey, 2013–14 and 2014–15; Research report; Australian Bureau of Agricultural and Resource Economics (ABARE): Canberra, Australia, 2015.

[2] 2. Kondo, N.; Monta, M.; Noguchi, N. Agricultural Robots: Mechanisms and Practice; Trans Pacific Press: Balwyn North Victoria, Australia, 2011.

[3] 3. Bac, C.W.; van Henten, E.J.; Hemming, J.; Edan, Y. Harvesting Robots for High-Value Crops: State-of-the-Art Review and Challenges Ahead. J. Field Robot. 2014, 31, 888–911.

[4] Nuske, S.T.; Achar, S.; Bates, T.; Narasimhan, S.G.; Singh, S. Yield Estimation in Vineyards by Visual Grape Detection. In Proceedings of the 2011 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS '11), San Francisco, CA, USA, 25–30 September 2011.

[5] Nuske, S.; Wilshusen, K.; Achar, S.; Yoder, L.; Narasimhan, S.; Singh, S. Automated visual yield estimation in vineyards. J. Field Robot. 2014, 31, 837–860.

[6] Yamamoto, K.; Guo,W.; Yoshioka, Y.; Ninomiya, S. On plant detection of intact tomato fruits using image analysis and machine learning methods. Sensors 2014, 14, 12191–12206.

[7] Wang, Q.; Nuske, S.T.; Bergerman, M.; Singh, S. Automated Crop Yield Estimation for Apple Orchards. In Proceedings of the 13th International Symposium on Experimental Robotics (ISER 2012), Québec City, QC, Canada, 17–22 June 2012.

[8] Bac, C.W.; Hemming, J.; van Henten, E.J. Robust pixel-based classification of obstacles for robotic harvesting of sweet-pepper. Comput. Electron. Agric. 2013, 96, 148–162.

[9] Hung, C.; Nieto, J.; Taylor, Z.; Underwood, J.; Sukkarieh, S. Orchard fruit segmentation using multi-spectral feature learning. In Proceedings of the 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Tokyo, Japan, 3–7 November 2013; pp. 5314– 5320.

[10] Kapach, K.; Barnea, E.; Mairon, R.; Edan, Y.; Ben-Shahar, O. Computer vision for fruit harvesting robots-state of the art and challenges ahead. Int. J. Comput. Vis. Robot. 2012, 3, 4–34.

[11] Seng, W.C., Mirisaee, S.H.: A new method for fruit recognition system. In: International Conference on Electrical Engineering and Informatics, ICEEI 2009, Selangor, Malaysia, pp. 130–134 (2009).

[12] Rocha, A., Hauagge, D.C., Wainer, J., Goldenstein, S.: Automatic fruit and vegetable classification from images. Computers and Electronics in Agriculture 70(1), 96–104 (2010).

[13] Aibinu, A.M., Salami, M.J.E., Shafie, A.A., Hazali, N., Termidzi, N.: Automatic fruits identification system using hybrid technique. In: Sixth IEEE International Symposium on Electronic Design, Test and Application (DELTA), Queenstown, pp. 217–221 (2011).

[14] Song, Y.; Glasbey, C.; Horgan, G.; Polder, G.; Dieleman, J.; van der Heijden, G. Automatic fruit recognition and counting from multiple images. Biosyst. Eng. 2014, 118, 203–215.

[15] Berns, R.S.: Principles of Color Technology, 3rd edn. Wiley, New York (2000).

[16] Group, T.M.R. Image Processing Toolbox User's Guide. 2008 [cited 14th November 2008]; Available from: http://www.mathworks.com/products/image/description1.html.

[17]https://medium.com/@sonish.sivarajkumar/k-nearest-neighbours-knn-algorithm-9900c1427726

[18] https://www.datacamp.com/community/tutorials/random-forests-classifier-python

[19] https://www.javatpoint.com/machine-learning-random-forest-algorithm

[20]https://medium.com/@raycad.seedotech/convolutional-neural-network-cnn-8d1908c010ab

[21] https://medium.com/@danqing/a-practical-guide-to-relu-b83ca804f1f7writ