2022-2023 Spring Semester

Final Report

Course: CS464

Title: Fruit Detector

Project Name: TuttuFruitti

Group Number: 1

Members:

Gülçin Özkahya	21903129
Vesile İrem Aydın	21902914
Ece Kahraman	21801879
Alper Mumcular	21902740

Table of Contents:

1. Introduction

Our collective interest was in image recognition, so we were looking for a fun way to implement this interest. In the end, we decided to choose fruit recognition as the topic of our project. This project aims to train our model to recognize fruits by their shapes, colors, and some other physical attributes. A dataset of fruit pictures is used for training and testing. We have chosen a dataset consisting of fruit images of 15 different types of fruits. Besides the dataset, including various kinds of fruits, the fruits also range in color and ripeness. In our project, we are targeting to classify fruits considering the name of fruits and also their ripeness and color, and we use the following machine learning models: Random Forest Classifier, Gaussian Naive Bayes Classifier, and Convolutional Neural Network (CNN).

2. Dataset

2.1 Dataset Split

For all models, we have decided to use only 20% of the original data set.

The written split code for Random Forest, Gaussian Naive Bayes, and Convolutional Neural Network is below:

```
dataset = img_df.iloc[0:14109]

X = dataset['pixel']

Y = dataset['label']

X_train, X_test, Y_train, Y_test = train_test_split(X, Y, train_size=0.8, stratify=Y, random_state=1)

dataset_test = img_df.iloc[67013:70540]
```

For all models, the data split was done as given below:

64% → Train

16% → Validation

20% → Test

2.2 Data Preprocessing

2.2.1 Random Forest

We have applied preprocessing to raw input to make the images more suitable for classification methods. As seen below, we have applied a resize function to the images. The purpose of this operation was to standardize the size of the images to 50x50 pixels. By applying resize, we made the images more comparable and reduced the computational complexity. In addition to resizing operations, we have used the “flatten” function to convert a 2D pixel array into a 1D array. Flatten operation simplifies the processing of arrays, and the representation of data becomes more understandable.

```
def img_to_array(img_path):
    return cv2.resize(cv2.imread(img_path), (50, 50)).flatten()

dataset['pixel'] = dataset['Filepath'].apply(img_to_array)
dataset.head()
```

In order to have the same scale for features, we have made feature scaling, using “StandardScaler” from Scikit-learn. By using this function, we make sure that the classification methods we use are not affected by the feature’s scale.

```
X_train = np.stack(X_train)
scaler = StandardScaler()
scaler.fit(X_train)
X_train_norm = scaler.transform(X_train)
X_test = scaler.transform(np.stack(X_test))
```

Additionally, we have used the “cvtColor” function from OpenCV. We have used this function to convert images from BGR(Blue Green Red) to RGB(Red-Green-Blue). We have applied this preprocessing step since a lot of libraries use the Red-Green-Blue format.

```
img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

Lastly, we have applied PCA to the data to simplify images and improve the effectiveness of the classification methods. For this purpose, we have extracted the top 500 principal components.

```
pca = PCA(n_components=500)
pca.fit(X_train)
```

2.2.2 Gaussian Naive Bayes

In order to train the model, first the images are reduced to 50x50 pixels. After resizing, the images are typically represented as a matrix of pixel values, with each pixel corresponding to a numerical value. After that, the augmented training set images are flipped vertically to increase the number of samples and improve the accuracy (also done in Random Forest). To make it easier to train a Gaussian Naive Bayes model, the matrix is flattened into a 1D array, which means that all the pixel values are concatenated into a single long vector.

It is important to normalize the pixel values in both the training and test sets to ensure that they are on a similar scale. The Sklearn StandardScaler is used for normalization, which subtracts the mean and scales the data to unit variance.

Finally, to further reduce the dimensionality of the data and make it easier to train a Gaussian Naive Bayes model, the training set is often subjected to principal component analysis (PCA). The training set is reduced from its original dimensionality to 500 dimensions, which explains more than 90% of the images while still reducing the computational overhead of the model.

2.2.3 Convolutional Neural Network

Since CNN doesn't want a 2D feature set like other models, the processes in other models were not applied. In this model, the size of all images in the existing dataset was changed to 50x50 and the model was trained. The reason we made the image size 50x50 was that if they had stayed at their original size, training the model would have taken very long hours, maybe days. So we made a tradeoff between image quality and the speed of training the model.

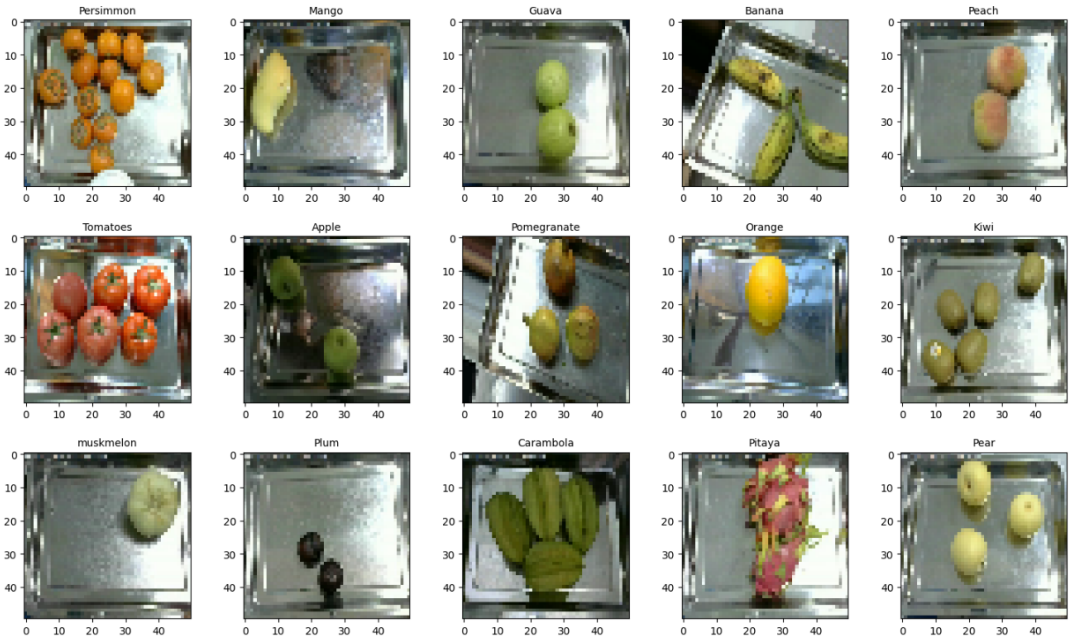

Figure 1: Preprocessed images in CNN

2.3 Detailed Analysis

For this part, we have prepared visual and numerical data about the dataset we have. First of all, there are 15 different fruit types in the dataset. These fruit types are as follows: *Apple, Banana, Carambola, Guava, Kiwi, Mango, Orange, Peach, Pear, Persimmon, Pitaya, Plum, Pomegranate, Tomato,* and *Muskmelon*. After determining the fruit varieties, we prepared a pie chart showing the percentage distribution of the fruits in the data set (see Figure 2).

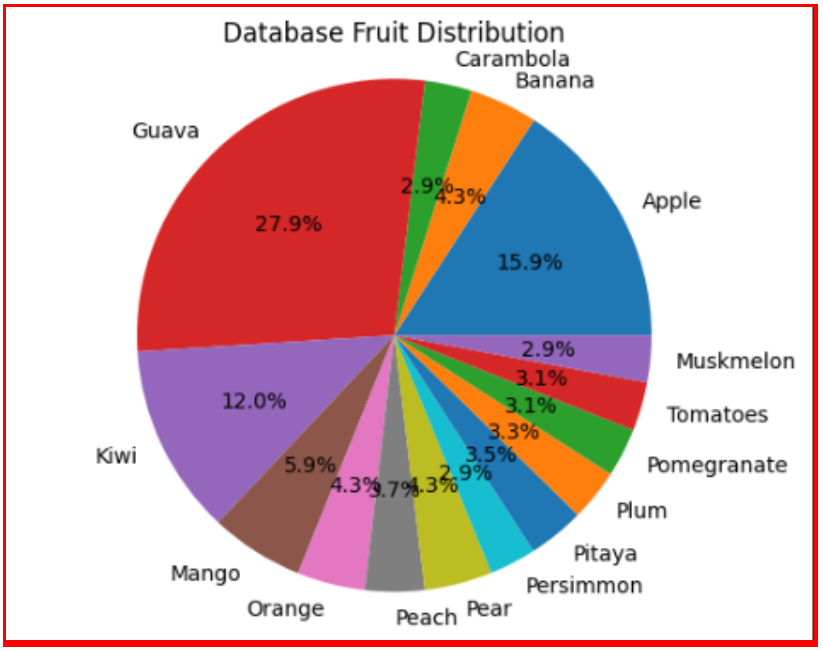

Figure 2: Database fruit distribution

As you can see, the number of Guava, Kiwi, and Apple is significantly higher than other fruits. Then, instead of the percentage distribution, we calculated how many pictures there are for each fruit.

```
Occurrence of each fruit in the database
fruit
Apple          11185
Banana          3027
Carambola       2080
Guava          19698
Kiwi            8465
Mango           4154
Orange          3012
Peach           2629
Pear            3012
Persimmon       2072
Pitaya          2501
Plum            2298
Pomegranate     2167
Tomatoes        2171
muskmelon       2078
```

Figure 3: Occurrence of each fruit in the database

3. Coding Environment Summary

During the implementation, we preferred to use Google Colaboratory. Colab is a platform that implements the read-eval-print loop with its code cells, which prevents us from running the whole code after every change [7]. Colab is also integrated with Google's services, the most relevant one to us was that our notebooks were automatically saved to our Google Drives and we were able to share and co-edit them. Colab comes with more libraries preinstalled than Jupyter Notebook. That is why we did not need to locally install the libraries we used like openCV, Keras, and scikit-learn. Its integration with Kaggle also prevented us from downloading the huge dataset.

```
!pip install kaggle

! mkdir ~/.kaggle

! cp kaggle.json ~/.kaggle/

! chmod 600 ~/.kaggle/kaggle.json

! kaggle datasets download chrisfilo/fruit-recognition

! unzip fruit-recognition.zip -d archive -p

! mkdir archive

! unzip fruit-recognition.zip -d archive
```

With the code above, we were able to quickly download and run the dataset in each session in Google Colab. For this code to work, we got a key named "kaggle.json" from Kaggle. Each time, we first uploaded this file into Google Colab and ran the code like that.

4. Details of Trained Models and Configuration

4.1. Random Forest Classification

Random Forest is one of the most frequently used machine learning models. It can be used in both regression and classification problems. In this algorithm, the output of the many decision trees is used to get a result for the problem [1].

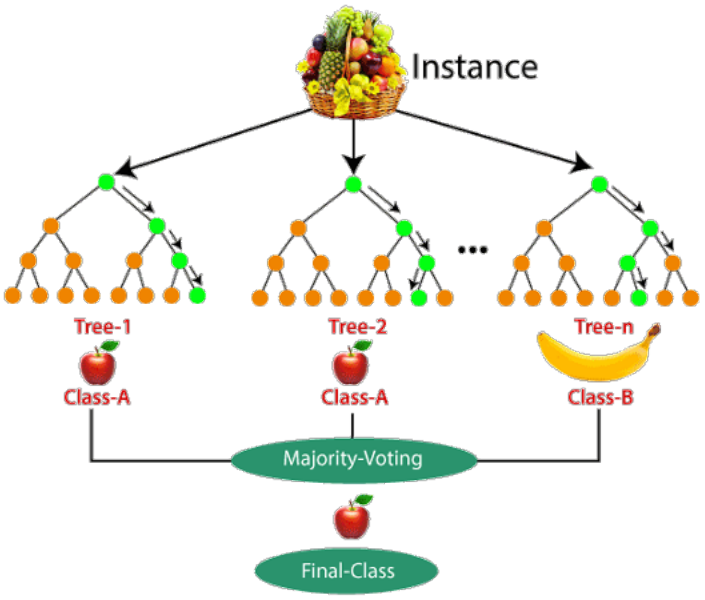

Figure 4: Visualization of how Random Forest Classifier works [2]

In the random forest classifier, every single tree includes the same nodes. In order to decide on a feature to split “impurity” or “purity” of the dataset is calculated. Impurity is given by calculating the Gini index. The feature that gives the lowest Gini index is chosen for the further split [3]. The Gini index is given as a result of the following calculations:

$$Gini\ Index = 1 - \sum_{i=1}^{n}\left(P_i\right)^2$$
$$= 1 - [(P_+)^2 + (P_-)^2]$$

Figure 5: Formula of the Gini index [3]

In the formula given above P_+ represents the positive class and P_- represents the negative class. We have chosen random forest as one of our models because even in large datasets, it gives high accuracy. Also, the training time in this algorithm is relatively less [4]. Finally, in this algorithm, overfitting is less likely because a lot of decision trees are combined, and by combining, bias and variance are reduced so the occurrence of overfitting is less likely.

4.1.1 Hyperparameter Tuning for Random Forest

We have accomplished hyperparameter tuning using GridSearchCV to make the Random Forest classifier more optimized. We have used GridSearchCV to search through different combinations of hyperparameters to pick the best one for the Random Forest model. We have used two hyperparameters for tuning in our code which are the following:

n_estimators: The number of decision trees in the random forest.
we have used n_estimator values: 125, 100, 75, 50, 25

max_depth: The maximum depth of the decision tree
we have used max_depth values: None, 10, 30

We have trained and evaluated our classifier on various combinations of hyperparameters. To retrieve the best hyperparameters, we have accessed the **best_estimator** attribute of the GridSearchCV class as the following:

```
# Best hyperparameters
print(grid_search.best_estimator_.n_estimators)
print(grid_search.best_estimator_.max_depth)
```

For the n_estimators we got 125, and for the max_depth we got None.

We created a graph to observe the relationship between the number of estimators (n_estimators) in a Random Forest classifier and the corresponding accuracy of the model. The aim of creating this graph is to understand how varying the number of estimators impacts the performance of the classifier.

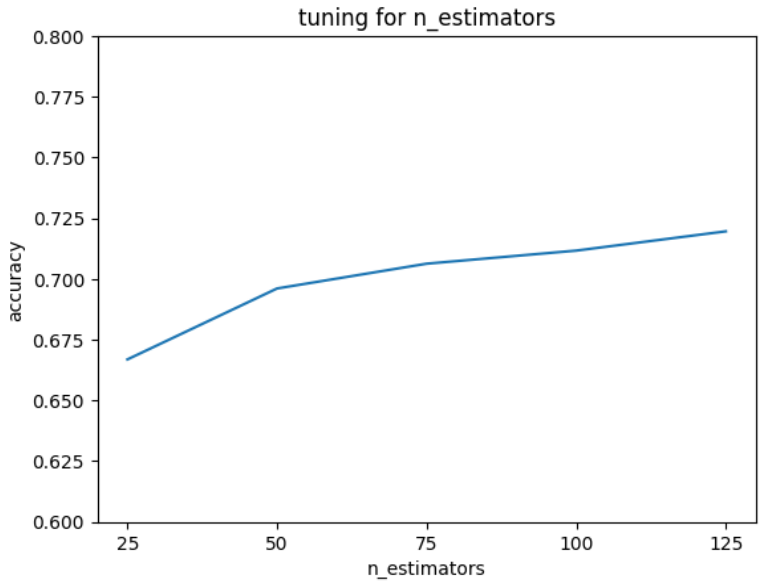

Figure 6: Hyperparameter Tuning for n_estimators of Random Forest

We observe that as the number of estimators increased, accuracy increased. Thus, as the estimator number increases, the classifier is able to capture the complex relationship, and the effect of individual decision trees is reduced. This has resulted in better accuracy. It must be noted that as the number of estimators increased it took longer to run the code. Hence, there should be a balance between the estimator number in other words model complexity and the computational efficiency.

4.1.2 Results of Random Forest

4.1.2.1 Performance Metrics

Performance Metrics of Random Forest Classifier:

	precision	recall	f1-score	support
Apple	0.44	0.96	0.61	569
Banana	0.98	0.33	0.49	135
Carambola	1.00	0.18	0.30	112
Guava	0.81	0.97	0.88	940
Kiwi	0.92	0.92	0.92	432
Mango	0.94	0.56	0.70	227
Orange	0.97	0.44	0.61	158
Peach	0.95	0.46	0.62	119
Pear	1.00	0.45	0.62	133
Persimmon	1.00	0.49	0.66	130
Pitaya	1.00	0.39	0.56	128
Plum	0.99	0.71	0.83	120
Pomegranate	1.00	0.52	0.69	111
Tomatoes	1.00	0.38	0.55	113
muskmelon	0.98	0.46	0.63	100
accuracy			0.73	3527
macro avg	0.93	0.55	0.64	3527
weighted avg	0.84	0.73	0.72	3527

Figure 7: Performance metrics of Random Forest

The accuracy of the Random forest model is 0.73. Accuracy is one of the most commonly used performance metrics to assess classification models, and it gives the proportion of correctly predicted instances over the total number of instances.

In the Random Forest Classifier case, accuracy which is a value of 0.73 shows that the random forest model predicted class labels correctly in 73% of the fruit photos. Only accuracy might not be enough to provide detailed information on the model's performance. Thus, precision, recall, and f1-score are also provided to depict the performance of the model.

The recall is poor for carambola(0.18), and banana(0.33) which can be due to the dataset used for training the model being imbalanced, meaning there is a significant difference in the number of instances of carambola and banana and others so the model may be biased towards the majority class. This can lead to a low recall for the minority class, as the model struggles to identify the positive instances. Precision is poor for apples which are 0 point forty-four due to false positives or in other words predicting fruits as apples while they are not.

4.1.2.2 Confusion Matrix of Random Forest

Table 1: Confusion matrix of Random Forest

The confusion matrix for the random forest given above provides information about the performance of the random forest model. The model has a tendency to label fruit images as apples because the FP number is high compared to other kinds of fruits. On the other hand, apple guava and kiwi images are correctly guessed since their TP values are high.

4.1.2.3 Example Predictions

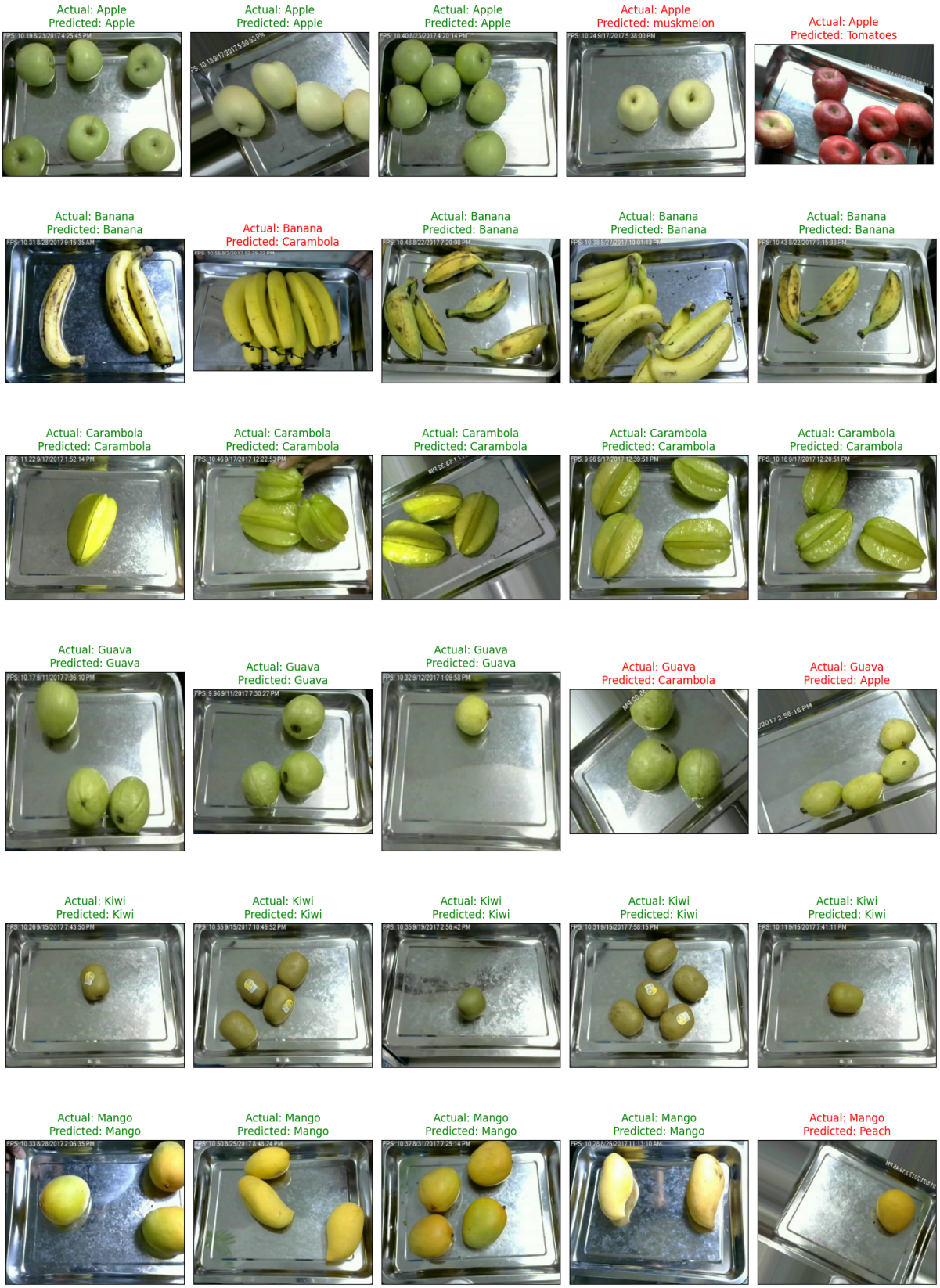

Figure 8: Some example predictions of Random Forest Classifier

4.2. Gaussian Naive Bayes Classification

In the Naive Bayes classification, features are assumed to be independent. Gaussian is a kind of Naive Bayes that includes normal Gaussian Distribution. Gaussian Naive Bayes rule is as the following:

$$P(X|Y) = \frac{P(X,Y)}{P(Y)} = \frac{P(Y|X)P(X)}{P(Y)}$$

$$P(x_i \mid y) = \frac{1}{\sqrt{2\pi\sigma_y^2}} \exp\left(-\frac{(x_i - \mu_y)^2}{2\sigma_y^2}\right)$$

Figure 9: Some required formulas for Gaussian Naive Bayes Classification [5]

In the predictions made using Naive Bayes the following calculations are made: (the number of features is assumed to be n, and the estimation type is assumed to be k)

$$P(Y_i = y_k|X_1, X_2, ..., X_n) = \frac{P(Y_i=y_k)P(X_1,X_2,...,X_n|Y_i=y_k)}{P(X_1,X_2,...,X_n)} \propto P(Y_i)P(X_1, X_2, ..., X_n|Y_i = y_k)$$

$$P(Y_i = y_k)\, P(X_1, X_2, ..., X_n|Y_i = y_k) = P(Y_i = y_k) \prod_{i=1}^{n} P(X_i, Y_i = y_k)$$

To perform computations in an easier manner log is taken.

$$log(P(Y_i = y_k) \prod_{i=1}^{n} P(X_i, Y)) = log(P(Y_i = y_k) + \sum_{i=1}^{n} log(P(X_i, Y))$$

In Naive Bayes, for each possible class, labels the likelihood is calculated using the above formula. For each conditional probability, the Gaussian distribution formula given in Figure 3 must be plugged in. Then the maximum of the likelihoods calculated is chosen as the label. We have chosen Gaussian Naive Bayes as one of our classification methods because it might be useful to classify fruit images based on the texture of the image and color distribution.

4.2.1 Hyperparameter Tuning for Gaussian Naive Bayes

Hyperparameter tuning is the process of finding the optimal values for the hyperparameters of the Gaussian Naive Bayes model. For this model, there are two main hyperparameters to tune: the priors and the regularization parameter (var_smoothing).

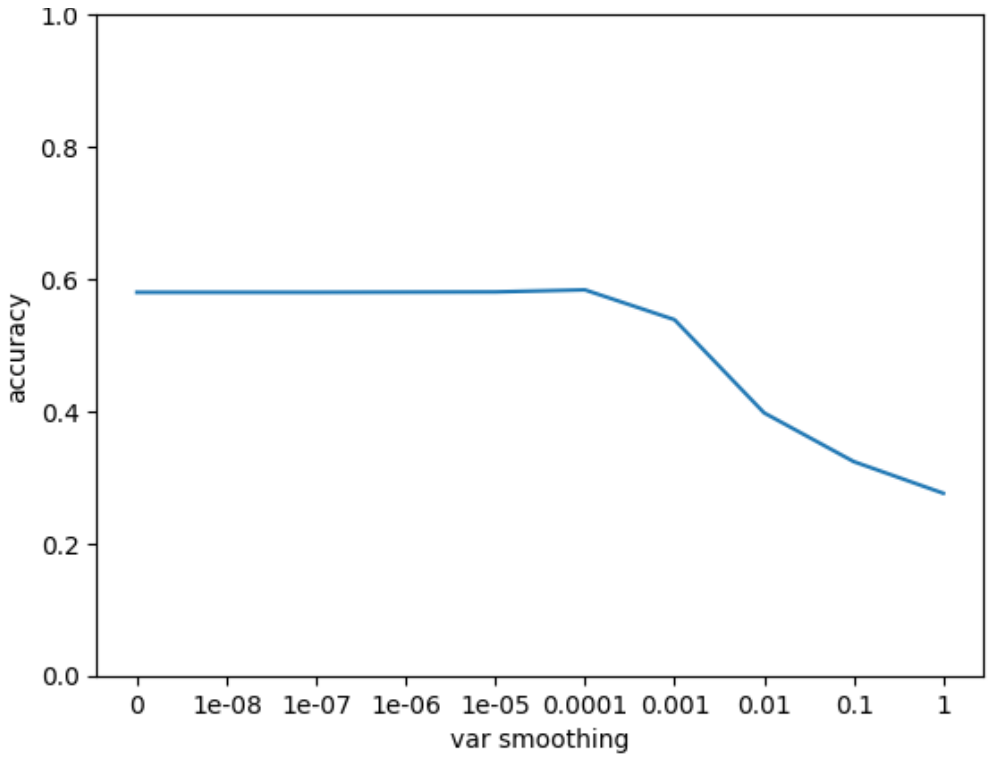

Figure 9: Hyperparameter Tuning for var_smoothing of Gaussian Naive Bayes

Gaussian Naive Bayes, the var_smoothing hyperparameter is used to add a small smoothing value to the variances of each feature in order to prevent the variance from being zero. This smoothing helps avoid mathematical issues, such as division by zero, during the calculation of the likelihood probabilities. The default value for var_smoothing is 1e-9. We experimented on different values: [1e-11, 1e-10, 1e-9, 1e-8, 1e-7, 1e-6, 1e-5, 1e-4, 1e-3, 1e-2, 1e-1, 1,0,1e-05], and find the optimum value 1e-3 for our model.

The priors hyperparameter allows you to specify the prior probabilities for each class in the Gaussian Naive Bayes algorithm. These priors represent your prior belief or knowledge about the distribution of classes in the dataset. By default, the priors parameter is set to None, which means the class priors are inferred from the training data. We experimented on different prior values for our model: priors: [None, uniform, weighted]. Uniform prior assigns the same prior value for each class, which is 1 / (number of classes). Weighted priors are proportional to the number of images

of each class in the validation dataset, for instance: (number of apples) / (number of all images). This approach creates a tendency towards the large classes and increases the number of labels for them. However, experiments showed that uniform and weighted priors perform poorly compared to None for the model.

4.2.2 Results of Gaussian Naive Bayes

4.2.1.1 Performance Metrics

Performance Metrics:

	precision	recall	f1-score	support
Apple	0.50	0.31	0.38	569
Banana	0.59	0.59	0.59	135
Carambola	0.69	0.61	0.65	112
Guava	0.67	0.83	0.74	940
Kiwi	0.62	0.57	0.60	432
Mango	0.41	0.48	0.44	227
Orange	0.78	0.61	0.69	158
Peach	0.25	0.24	0.24	119
Pear	0.35	0.43	0.38	133
Persimmon	0.97	0.52	0.68	130
Pitaya	0.92	0.69	0.79	128
Plum	0.98	0.68	0.80	120
Pomegranate	0.72	0.58	0.64	111
Tomatoes	0.29	0.63	0.39	113
muskmelon	0.35	0.39	0.37	100
accuracy			0.58	3527
macro avg	0.61	0.54	0.56	3527
weighted avg	0.61	0.58	0.58	3527

Figure 10: Performance metrics of Gaussian Naive Bayes

The accuracy of the Gaussian Naive Bayes model is 0.58. Accuracy provides a general measure of the model's predictive power but may not be sufficient in all cases. The Performance Metrics table illustrates precision, recall, and f1-score values for all classes to provide a more comprehensive measure of the model. As can be seen in the table, precision is poor for peach (0.25) and muskmelon (0.35) while recall is poor for peach(0.24), and apple (0.31).

4.2.1.2 Confusion Matrix of Gaussian Naive Bayes

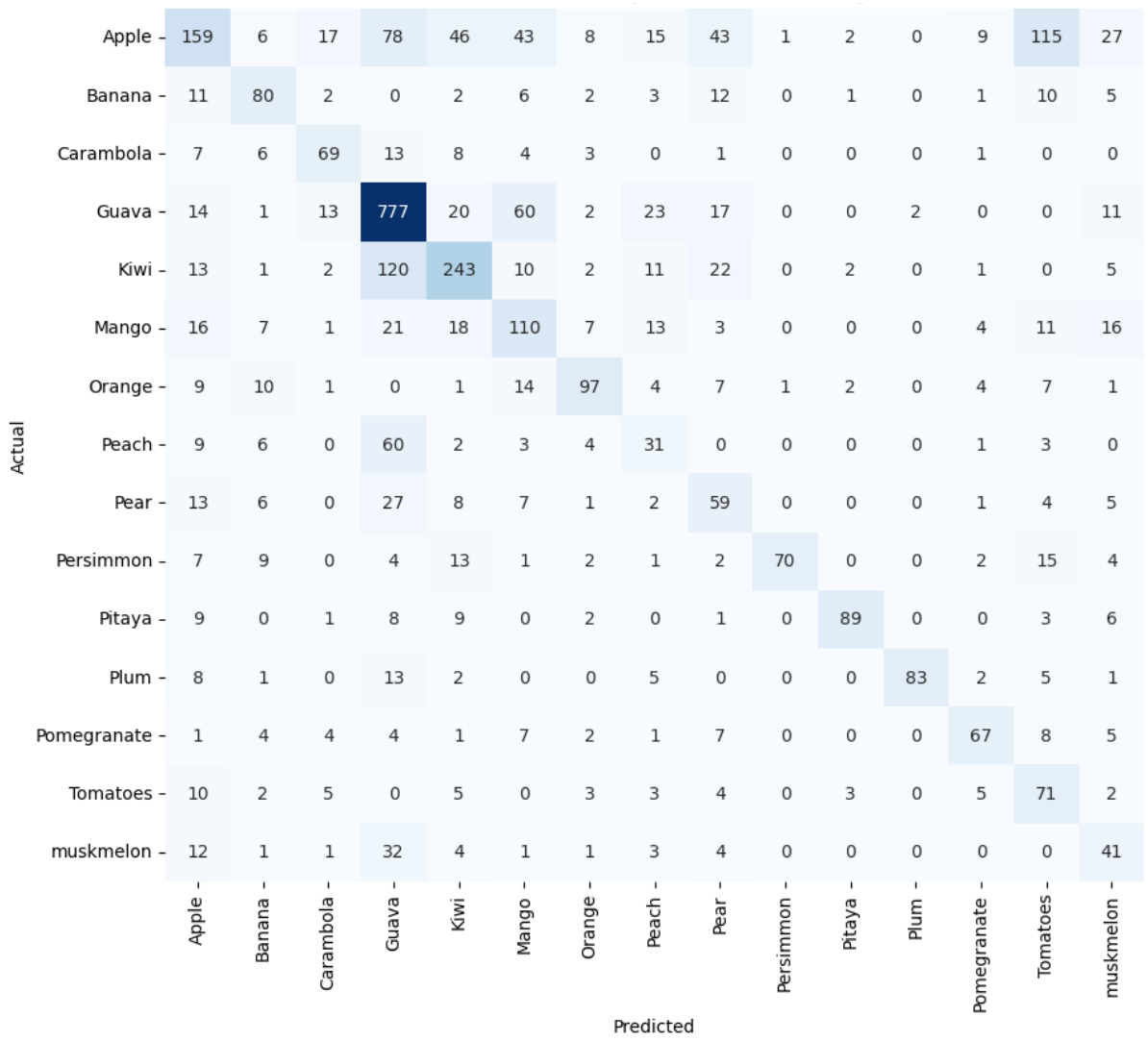

Table 3: Confusion matrix of Gaussian Naive Bayes

The confusion matrix can help evaluate the model's performance in terms of true positives, true negatives, false positives, and false negatives for each class. In this model, FN values are high for apples and TP values are high for guava. The dataset is skewed towards guava and this can be the reason for high TP values for guava. Apple images are likely to be classified as guava or tomato which may be because of the imbalanced dataset.

4.2.1.3 Example Predictions

Figure 11: Some example predictions of Gaussian Naive Bayes Classifier

4.3 Convolutional Neural Network Classification

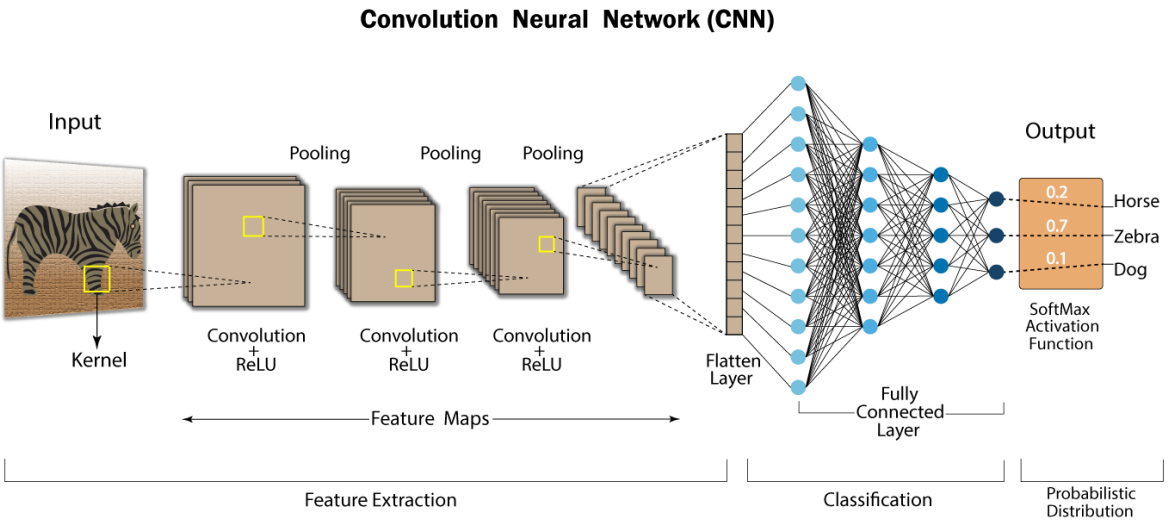

Figure 12: Structure of CNN [6]

For CNN, we used the Keras library in the project. In our project, we have a CNN structure as follows (Feature Extraction, Classification(including Probabilistic Distribution)):

Convolution (32 filters) + ReLU

Pooling

Convolution (64 filters) + ReLU

Pooling

Convolution (64 filters) + ReLU

Pooling

Convolution (64 filters) + ReLU

Pooling

Flattening

Dense (Count: 256), Activation: ReLU, Dropout: 0.25

Dense (Count: 15), Activation: Softmax

Loss = Categorical Crossentropy

Optimizer = Adam

We chose to use Softmax because it is more suitable for classification problems than the Sigmoid function.

We also used 2 callbacks:

1- EarlyStopping: If there is no improvement then it will stop doing more epochs. This is used to prevent overfitting.

2- ModelCheckpoint: It chooses the most accurate epoch run.

4.3.1 Hyperparameter Tuning for CNN

We focused on tuning the batch-size hyperparameter for the CNN model. Because we used two callback methods while model fitting, we did not need to tune for the epoch_num parameter.

The batch sizes to test were chosen from 8 to 256, being doubled on each step. The value accuracy results are given in the table below:

Batch Size	Value Accuracy
8	0.9316
16	0.9334
32	0.9263
64	0.9415
128	0.9309
256	0.9054

Previously, we had chosen the batch size as 16, and it resulted in a value accuracy of 0.92. While 16 resulted in 93% value accuracy this time, the highest value accuracy was 0.9415. Hence, we continued our experiment with the batch size chosen as 64.

4.3.2 Results of CNN

4.3.2.1 Accuracy - Value Accuracy

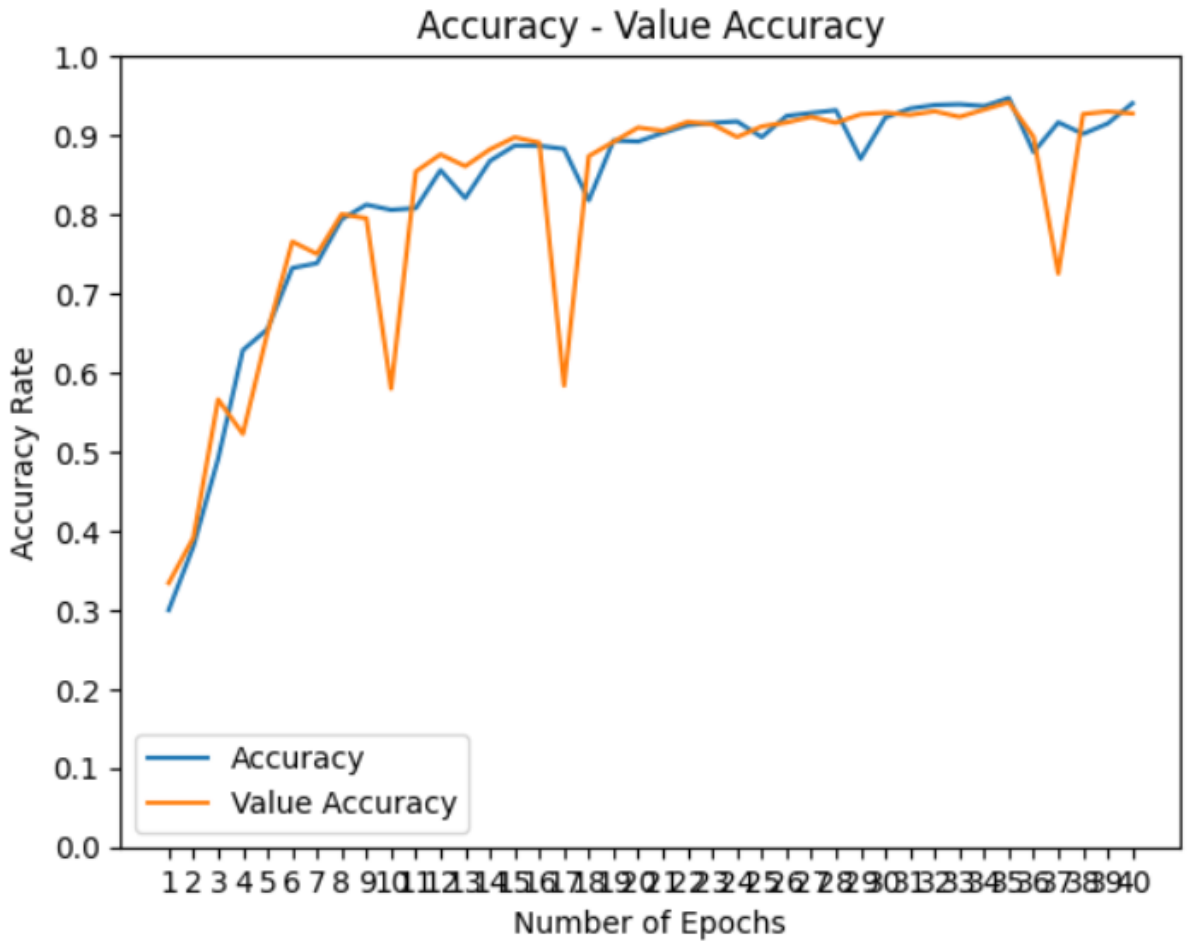

Figure 13: Accuracy - value accuracy graph

When we look at the graph, we can generally observe that accuracy and value accuracy increase as the number of epochs increases. Value accuracy drastically declines during some epochs. Accuracy itself also declines during some epochs but it is not as sharp as the previous one. If we were to take the average of these two values, accuracy would be higher because of the sharp declines the value accuracy experiences. Aside from those epochs with drastic reductions, value accuracy follows the same trend as the accuracy. You will also see that the epoch count ends at 40. This is because of the callbacks used.

4.3.2.2 Performance Metrics

Performance Metrics:

	precision	recall	f1-score	support
Apple	0.91	0.92	0.92	576
Banana	0.79	0.83	0.81	151
Carambola	0.99	0.95	0.97	104
Guava	0.99	0.99	0.99	938
Kiwi	0.98	0.99	0.99	472
Mango	0.90	0.93	0.92	213
Orange	0.96	0.98	0.97	160
Peach	0.94	0.93	0.94	134
Pear	0.86	0.96	0.91	154
Persimmon	0.97	0.93	0.95	111
Pitaya	0.96	0.92	0.94	107
Plum	0.99	0.97	0.98	110
Pomegranate	0.99	0.77	0.87	102
Tomatoes	0.93	0.96	0.94	117
muskmelon	0.84	0.74	0.79	78
accuracy			0.94	3527
macro avg	0.93	0.92	0.92	3527
weighted avg	0.95	0.94	0.94	3527

Figure 14: Performance metrics of CNN

The overall accuracy before the batch size tuning was 0.92. After the tuning, it climbed to 0.94, an increase of 2.2%. Both macro and weighted averages remain above 0.90. As you can see in the figure above, most of the recall and precision values are above 80% to 90%. However, we have a few exceptions. These are banana and muskmelon. It is striking that these two also have a low f1 score value. Although we cannot predict exactly why these are caused, we think that it may be due to the low numbers in the test dataset compared to other ones.

4.3.2.3 Confusion Matrix

Table 5: Confusion matrix of CNN

The model mostly makes accurate predictions. The results of the confusion matrix also indicate that the model avoids overfitting the data. Interestingly, the false positive number is the highest in the apple set. This could be explained as the data set is skewed towards the apple set. Interestingly, false positives in bananas are also high, but because the number of bananas is lower, the precision of bananas decreases. On the other hand, the number of false negatives of the muskmelon set catches attention. Because the number of muskmelon images is quite low, it decreases the recall metric of the set. Pomegranates also suffer from the same condition as muskmelon.

4.3.2.4 Example Predictions

Figure 15: Example predictions of CNN

As you can see, it predicts with a high accuracy rate, although there are a few incorrect predictions.

5. Work Done

- Data preprocessing
- Detailed Analysis of the dataset
- Implementing Random Forest Classifier
- Implementing Gaussian Naive Bayes Classifier
- Implementing Convolutional Neural Network Classifier
- Preparing performance metrics and confusion matrix of each model
- Experiments with different hyperparameters
- Documentation of the codes.
- Adding validation set for Random Forest and Naive Bayes classifiers

6. Comparison and Conclusion

	Gaussian NB	Random Forest	CNN
Accuracy	0.58	0.73	0.94
Macro Precision	0.61	0.93	0.93
Macro Recall	0.54	0.55	0.92
Weighted Prec.	0.61	0.84	0.95
Weighted Recall	0.58	0.73	0.94

Table 6: The comparison between model performance metrics

There is an increasing trend in the performance metrics of the models, from Gaussian NB to CNN. This indicates that Gaussian NB is not well suited for image classification problems. Gaussian NB treats features as true or false, and it assumes every feature is completely independent of the other. However, in an image, not every feature is independent. That is why Gaussian NB loses information about the relationships between features, dimensions, and spatiality of images. Still, it performs better on positive predictions as it has higher macro precision and weighted precision. On the other hand, it suffers in recall metrics because of the high number of false predictions.

Random Forest (RF) performs much better than Gaussian NB but much worse than CNN. Because RF implements decision trees, it is able to be better at handling complex problems. In each decision split for each feature, RF conserves the spatiality, the dimensions, and the relationships between them; it does not assume independence, unlike Gaussian NB. RF is also better in precision metrics, but the macro precision metric is curiously very high. The reason could be that the multiple decision trees RF creates for a prediction and then aggregating them. If there are multiple trees with very high outcomes, then the aggregated result will also be high.

CNN performed the best. It was not a surprise because neural networks are known for being very good at handling complex data and problems. They are also at the core of the currently popular network models. The performance metrics of our CNN model are also very similar to each other, which indicates that the model did not overfit the data and performed in balance overall.

7. Difficulties Encountered

The first difficulty encountered was how to preprocess the images for Gaussian NB. We had to learn and understand how to implement Random Forest and CNN models. During the project, we had a falling out with one of our group members, so we had to rebalance the workload between the remaining four of us.

8. Division of Work

Alper Mumcular: Contributed to data preprocessing, data analysis, training of the CNN model, and preparing visual elements of the trained models.

Ece Kahraman: Contributed to the data analysis and the training of the CNN model, and to preparing visual elements.

Gülçin Özkahya: Contributed to data preprocessing, data analysis, and training of the RFC model.

Vesile İrem Aydın: Contributed to data preprocessing, preprocessing of data, and training of the Naive Bayes model.

9. References

1- “What is Random Forest?”, *IBM Cloud Learn Hub*. Available: https://www.ibm.com/cloud/learn/random-forest (Accessed: Apr. 20, 2023).

2- “Understand Random Forest Algorithms With Examples”, *Analytics Vidhya*, Available:https://www.analyticsvidhya.com/blog/2021/06/understanding-random-forest/ (Accessed: Apr. 20, 2023).

3- “An Introduction to Random Forest Algorithm for Beginners”, *Analytics Vidhya*, Available: https://www.analyticsvidhya.com/blog/2021/10/an-introduction-to-random-forest-algorithm-for-beginners/ (Accessed: Apr. 21, 2023)

4- “Random Forest Algorithm”, *javatpoint*, Available: https://www.javatpoint.com/machine-learning-random-forest-algorithm (Accessed: Apr. 22, 2023).

5- ”Gaussian Naive Bayes”, *OpenGenius IQ*, Available: https://iq.opengenus.org/gaussian-naive-bayes/ (Accessed: Apr. 22, 2023).

6- ”Basics of CNN in Deep Learning”, *Analytics Vidhya*, Available: https://www.analyticsvidhya.com/blog/2022/03/basics-of-cnn-in-deep-learning/ (Accessed: Apr. 22, 2023).

7- J. Horcasitas. “What is REPL?”, *DigitalOcean.* Available: https://www.digitalocean.com/community/tutorials/what-is-repl (Accessed: Apr. 24, 2023).