DEPARTMENT OF DIGITAL SYSTEMS

NCSR DEMOKRITOS
INSTITUTE OF INFORMATICS AND
TELECOMMUNICATIONS

Fruit Quality Classifier using Machine Learning

Ηλίας Αλεξανδρόπουλος
mtn2302

Σπυρίδων Γεωργούλας
mtn2309

Βασιλική Ρέντουλα
mtn2317

Abstract

Machine Learning algorithms are getting engaged in more and more industries nowadays. Specifically in argiculture, a common problem that usually comes up is the inspection of the quality of the products. This inspection is typically performed by a human being, a work that requires a lot of time to be performed effectively. Machine learning-based fruit quality classification is a novel strategy that will transform the agriculture sector. Machine learning (ML) is used in this industry to automate and enhance the accuracy of produce quality identification. In this project, a SVM model is proposed, capable of classifying fruits based on their type and their quality through image data. For our task we are using Support Vector Machine (SVM) models. We choose SVMs for their effectiveness in high-dimensional spaces and their ability to handle non-linear data through the use of kernel functions. We specifically test four variations of SVMs kernels: Linear, Radial Basis Function (RBF), Polynomial, Sigmoid, to ascertain their performance across our multi-class classification problem.

Contents

Chapter 1

Introduction

1.1 The Fruit Classification Problem

Machine Learning is a technology that is getting engaged in more and more industries nowadays. Specifically in argiculture, a common problem that usually comes up is the inspection of the quality of the products. This inspection is typically performed by a human being, a work that requires a lot of time to be performed effectively. Machine learning-based fruit quality classification is a novel strategy that will transform the agriculture sector. Machine learning (ML) is used in this industry to automate and enhance the accuracy of produce quality identification.

To do this, ML models must be trained on datasets containing characteristics like size, color, and shape. The purpose is to reduce manual inspection and increase productivity. Farmers and wholesalers may maximize resources and satisfy consumer demand for premium fruits by utilizing machine learning to assure uniform quality standards across their produce. Furthermore, machine learning algorithms are flexible and dynamic, which allows them to refine their accuracy and increase the overall effectiveness of fruit quality assessment over time. In the end, this innovation increases industry profitability and customer satisfaction by streamlining the supply chain and minimizing food waste by making it easier for consumers to receive premium fruits.

In this project, a SVM model is proposed, capable of classifying fruits based on their type and their quality through images data. For our task we are using Support Vector Machine (SVM) models. We choose SVMs for their effectiveness in high-dimensional spaces and their ability to handle non-linear data through the use of kernel functions. We specifically test four variations of SVMs kernels: Linear, Radial Basis Function (RBF), Polynomial, Sigmoid, to ascertain their performance across our multi-class classification problem.

Chapter 2

Literature Review

2.1 Support Vector Machines

Support Vector Machines (SVMs) constitute a family of supervised learning techniques utilized for classification and regression tasks. Conceived by Vladimir Vapnik and his colleagues in 1992, SVMs trace their conceptual roots back to earlier work, notably dating back to the 1960s, and are grounded in the Theory of Statistical Learning, representing an advancement from Perceptron algorithms [6].

In recent times, leveraging the exponential growth in computational expertise and storage capabilities, SVMs have found significant utility in various domains such as handwriting recognition, text categorization, and gene expression data classification.

A key advantage of SVMs lies in their ability to discern a hyperplane that distinctly separates positive and negative examples within the example space. This hyperplane is carefully chosen to maximize its distance from the nearest positive and negative instances, hence termed the maximum margin hyperplane. Consequently, an SVM can be characterized as an algorithm that identifies the hyperplane with the widest margin from the nearby examples that lie unclassified by it. Ultimately, the decision function of an SVM yields values within the range $[-1,+1][-1,+1]$, analogous to other hyperplanes.

The distinctive feature setting SVMs apart is their utilization of kernel functions. These functions enable the representation and computation of intricate relationships, employing a mechanism to transform and merge original features. This transformation renders a complex problem linear, facilitated by a four-dimensional vector of weights known as a feature vector (or support vector). Each kernel function is designed to adeptly transform and merge original features, thereby unveiling hidden structural patterns within the data.

Let's begin by examining the fundamental concept of Support Vector Machines (SVMs). When presented with a set of d-dimensional vectors, a linear classifier aims to delineate them using a $(d-1)$-dimensional hyperplane. Numerous hyperplanes could potentially

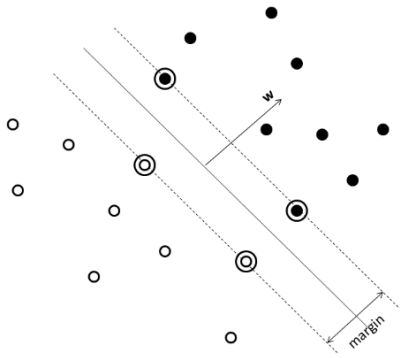


Figure 2.1: Visualization of the margin and support vectors in linearly separable data

classify the data. Introducing the concept of "margin," which denotes the distance between the closest samples on either side of the hyperplane, SVMs are crafted to select the hyperplane that maximizes this margin between the two classes. This optimal hyperplane, if it exists, is termed the maximum-margin hyperplane, and the linear classifier it forms is referred to as a maximum margin classifier. Subsequent sections will delve into the foundational principles of SVMs across various scenarios [1].

Notations: To describe the task in mathematical terms, we introduce the following notations:

- an example data point is denoted by $x \in \mathbb{R}^d$,
- class membership for a data point is denoted by $y \in \{-1, +1\}$,
- the set of training examples is denoted by $X = \{x_1, \ldots, x_n\}$,
- class labels for the training set is denoted by $Y = \{y_1, \ldots, y_n\}$.

2.1.1 Linear Support Vector Machines: Hard-Margin Case

At first, let's see the case when the data can be linearly separable. From the training data, we would like to learn a classification function $F : \mathbb{R}^d \to \{-1, +1\}$, a function that decides the class of a given example x.

In SVMs, F is chosen as the sign of a linear function i.e.,

$$F = \text{sign}(f(\mathbf{x}))$$

where $f : \mathbb{R}^d \to \mathbb{R}$ is the decision function. This decision function is based on the hyperplane separating the two classes. Any hyperplane separating the two classes will be of the form:

$$w^1 x_1^1 + w^2 x_2^2 + \cdots + w^d x_d^d + b = 0$$

where $x^1, \ldots, x^d$ are the components of x, and $w^1, \ldots, w^d$ are the coefficients of the weight vector $\mathbf{w}$ and b is the bias. Therefore, decision function f can be written as

$$f(\mathbf{x}) = (\mathbf{w}, \mathbf{x}) + b, \text{ where } (\cdot, \cdot) \text{ denotes the dot product}$$

The points $\mathbf{x}_i$ lying on the hyperplane satisfy $(\mathbf{w}, \mathbf{x}_i) + b = 0$, where $\mathbf{w}$ represents the normal vector to the hyperplane, $\frac{b}{\|\mathbf{w}\|}$ denotes the perpendicular distance from the hyperplane to the origin, and $\|\mathbf{w}\|$ is the Euclidean norm of $\mathbf{w}$. Let d^+ and d^- denote the shortest distance from the separating hyperplane to the closest positive and negative examples, respectively. The *margin* of a separating hyperplane is then given by $d^+ + d^-$. In the case of linear separability, the support vector algorithm aims to find the separating hyperplane with the maximum margin. Since the data is linearly separable, two hyperplanes can be chosen such that no points lie between them, maximizing the distance between them. These hyperplanes can be expressed as follows:

$$(\mathbf{w}, \mathbf{x}) + b = 1$$

$$(\mathbf{w}, \mathbf{x}) + b = -1$$

The margin which is the distance between those hyperplanes will be equal to $\frac{2}{\|\mathbf{w}\|}$. As we want to maximize the margin, we want to minimize the term $\|\mathbf{w}\|$. It would be difficult to solve $\min \|\mathbf{w}\|$ because of the square root involved in calculation of $\|\mathbf{w}\|$. Therefore, $\min \frac{1}{2}\|\mathbf{w}\|^2$ is used as the optimization problem to make it easier. Also, as we do not want any data points falling into the margin, we add the following constraints

$$(\mathbf{w}, \mathbf{x}_i) + b \geq 1 \quad \forall i \text{ with } y_i = 1$$

$$(\mathbf{w}, \mathbf{x}_i) + b \leq -1 \quad \forall i \text{ with } y_i = -1$$

After combining the above inequality constraints into a single constraint, we will have the following primal optimization problem. This is a quadratic programming problem.

$$\min_{\mathbf{w},b} \frac{1}{2}\|\mathbf{w}\|^2$$

subject to

$$y_i((\mathbf{w}, \mathbf{x}_i) + b) \geq 1$$

Note that factor $\frac{1}{2}$ is used for analytical convenience.

This issue is resolved through the application of traditional quadratic programming approaches. Alternatively, one can address its dual form, which is more straightforward and leads to solutions that match those obtained from the primal problem. Since the primal optimization task is convex, the solution it yields is singular, albeit the α_i coefficients may differ. Instances where α_i is nonzero are identified as support vectors [5].

2.1.2 Linear Support Vector Machines: Soft-Margin Case

Up to this point, we've outlined the methodology behind SVMs when the training dataset exhibits linear separability. However, when the data doesn't lend itself to linear separation, the hard-margin SVM becomes infeasible. This section delves into the adaptation of hard-margin SVMs to address such inseparable cases. To penalize errors, non-negative slack variables ξi ≥ 0, where i = 1, . . . , n, are introduced as follows:

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, n \tag{2.1}$$

$$\xi_i \geq 0 \quad \forall i \tag{2.2}$$

For any training data point x_i , if $0 < \xi_i < 1$, it signifies that the data points don't maximize the margin but are still correctly categorized. However, when $\xi_i \geq 1$, it indicates misclassification by the optimal hyperplane. Hence, $\sum_i \xi_i$ serves as an upper limit on the count of training errors. To impose additional penalties for these errors, the objective function to minimize will be:

$$Q(\mathbf{w}, b, \xi) = \frac{1}{2}\|\mathbf{w}\|^2 + C\left(\sum_i \xi_i\right) \tag{2.3}$$

subject to

$$y_i(\mathbf{w} \cdot \mathbf{x}_i + b) \geq 1 - \xi_i \quad \text{for } i = 1, \ldots, n \tag{2.4}$$

In the provided equations, C represents the margin parameter, a value to be selected by the user. It governs the balance between maximizing the margin and minimizing classification errors. This constitutes a convex programming problem for any positive integer k. For $k = 2$ and $k = 1$, it also qualifies as a quadratic programming problem. Opting for $k = 1$ additionally offers the benefit that neither the ξ_i terms nor their corresponding Lagrange multipliers appear in the dual optimization problem.

We call the obtained hyperplane the soft-margin hyperplane. When $k = 1$, we call the support vector machine as the L1 soft-margin support vector machine and when $k = 2$, the L2 soft-margin support vector machine. First we shall discuss L1 soft-margin support vector machines. Similar to the case of separable case, at first lagrangian multipliers are introduced in the optimization function as follows

$$L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \beta) = \frac{1}{2}\|\mathbf{w}\|^2 + C\sum_{i=1}^{n}\xi_i - \sum_{i=1}^{n}\alpha_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] - \sum_{i=1}^{n}\beta_i\xi_i \tag{2.5}$$

where $\alpha_i, \beta_i \geq 0$ are non-negative Lagrangian multipliers. For optimal solution, the following Karush-Kuhn-Tucker(KKT) conditions should be satisfied

$$\frac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \beta)}{\partial w} = 0 \tag{2.6}$$

$$\frac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \beta)}{\partial b} = 0 \tag{2.7}$$

$$\frac{\partial L_P(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \beta)}{\partial \xi} = 0 \tag{2.8}$$

$$\alpha_i[y_i(\mathbf{w} \cdot \mathbf{x}_i + b) - 1 + \xi_i] = 0 \text{ for } i = 1, \ldots, n \tag{2.9}$$

$$\beta_i \xi_i = 0 \text{ for } i = 1, \ldots, n \tag{2.10}$$

$$\alpha_i \geq 0, \beta_i \geq 0, \xi_i \geq 0 \text{ for } i = 1, \ldots, n \tag{2.11}$$

Applying Equations (2.19) to (2.21) on Equation (2.18), we have the following

$$\mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i \tag{2.12}$$

$$\sum_{i=1}^{n} \alpha_i y_i = 0 \tag{2.13}$$

$$\alpha_i + \beta_i = C \text{ for } i = 1, \ldots, n \tag{2.14}$$

Substituting, the above, we obtain the following dual problem,

$$L_D(\boldsymbol{\alpha}) = \max \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j y_i y_j \langle \mathbf{x}_i, \mathbf{x}_j \rangle \tag{2.15}$$

subject to the constraints:

$$0 \leq \alpha_i \leq C, \tag{2.16}$$

$$\sum_i \alpha_i y_i = 0 \text{ for } i = 1, \ldots, n, \tag{2.17}$$

Not that the main difference now is that the α_i is now bounded by C. Our decision function will become

$$f(\mathbf{x}) = \sum_{i=0}^{n} \alpha_i y_i \langle \mathbf{x}_i, \mathbf{x} \rangle + b \tag{2.18}$$

where n_V is the number of training samples out of which, the feature vectors for $\mathbf{x}_i$, for which α_i is non-zero are called support vectors [5].

2.1.3 Non-Linear Support Vector Machines

Up until this point, our examination has focused on the construction of a large-margin hyperplane and its advantageous generalization properties. Yet, a substantial limitation exists, as our procedures to date have been linear with respect to the dataset. To adopt decision boundaries of a more general nature, the input dataset $\{x_1, \ldots, x_n\} \in X$ is projected into a high-dimensional feature space through a non-linear transformation $\psi : \mathbb{R}^d \to F$. In this elevated dimensional space, a linear classifier is determined. The essential criterion for the space F is the capability to define a dot product therein. This space may be of infinite dimensions, and no presumptions are made regarding the dimensionality of F.

Subsequently, our discussions have revolved around the formulation of a large-margin hyperplane and its pronounced ability for generalization. However, the linear disposition of our methods in relation to the data presents a notable constraint. To enable the application of more sophisticated decision boundaries, the input collection $\{x_1, \ldots, x_n\} \in X$ undergoes a transformation into a higher-dimensional feature realm via a nonlinear mapping $\psi : \mathbb{R}^d \to F$. Within this enhanced dimensional realm, the objective is to locate a linear classifier. The primary requisite for the space F is the establishment of a dot product, notwithstanding its potential to encompass infinite dimensions, with no explicit assumptions about the dimensionality of F.

This version rephrases the original text while keeping all mathematical expressions exactly the same.
For a given training data set, SVM is now constructed in F instead of $\mathbb{R}^d$, i.e., using the set of examples $\{\psi(x_1), y_1\}, \ldots, \{\psi(x_n), y_n\} \in \mathbb{R}^N \times \{\pm 1\}$.

Utilizing the transformed set of samples, our task is to approximate the decision function within F. It's naturally inferred that the complexity of developing a decision function in the input space escalates with the increase in pattern dimensions, a phenomenon often referred to as the curse of dimensionality.

So now, if we substitute x with $\psi(x)$ in Equations (2.28) and (2.31), our optimization function will be

$$\max_{\alpha} \left\{ \sum_{i=1}^{n} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j y_i y_j \langle \psi(x_i), \psi(x_j) \rangle \right\} \quad (2.33) \qquad (2.19)$$

and decision function will be

$$f(x) = \sum_{i=1}^{n} \alpha_i y_i \langle \psi(x_i), \psi(x) \rangle + b \quad (2.34) \qquad (2.20)$$

Note that the above equations require only the dot product of the feature vectors in the transformed high-dimensional space. These expensive calculations are reduced significantly by using "kernel-trick". Computation of the feature-map is bypassed by using

a kernel function k which results in the dot product of data points in the transformed space [5].

2.1.4 Kernels

In an alternate approach, rather than transforming the input vectors non-linearly and subsequently performing dot products with support vectors in the high-dimensional space F, the sequence of operations is reversed [5]. Initially, a comparison is made between two vectors in the input space, followed by a non-linear transformation of this comparison result, which could be through their dot product or a certain measure of distance. Training a non-linear SVM, which necessitates the computation of dot products $\langle \psi(x_i), \psi(x_j) \rangle$ in the transformed space, can be simplified by introducing an appropriate kernel function k, such that

$$k(x_i, x_j) = \langle \psi(x_i), \psi(x_j) \rangle$$

By constructing a matrix Q, such that $(Q)_{ij} = y_i y_j k(x_i, x_j)$ that will be used for the optimization, the decision function is written as:

$$f(x) = \sum_{i=1}^{n_{\text{SV}}} \alpha_i y_i k(\psi(x_i), \psi(x)) + b$$

The Kernel Trick

The critical inquiry then becomes identifying the function k that equates to a dot product within a certain feature space F. Put differently, the challenge is to discover a mapping ψ such that the kernel function k executes the dot product in the space delineated by ψ. Mercer's theorem provides clarity to this query. Prior to delving into Mercer's theorem, it is essential to consider the subsequent definitions.
Let X be a non-empty set. A function $k : X \times X \to \mathbb{R}$ is called a positive definite kernel function if:

- k is symmetric, i.e., $k(x, y) = k(y, x)$ for all $x, y \in X$.
- For any finite set of points $x_1, \ldots, x_n \in X$, the kernel matrix $K_{ij} = (k(x_i, x_j))_{i,j}$ is positive semi-definite, i.e., for all vectors $t \in \mathbb{R}^n$:

$$\sum_{i,j=1}^{n} t_i K_{ij} t_j \geq 0$$

Hilbert Space

A vector space H is called a Hilbert space if it is equipped with an inner product $\langle \cdot, \cdot \rangle_H : H \times H \to \mathbb{R}$ and it is complete under the induced norm $\|v\|_H = \sqrt{\langle v, v \rangle_H}$, i.e., all Cauchy sequences of elements in H converge to a limit that lies in H.

Mercers Theorem

Let X be a non-empty set. For any positive definite kernel function $k : X \times X \to \mathbb{R}$, there exists a Hilbert space H and a feature map $\psi : X \to H$ such that

$$k(x, y) = \langle \psi(x), \psi(y) \rangle_H$$

where $\langle \cdot, \cdot \rangle$ denotes the inner product in H. For a kernel function k to be valid, the completeness property is of less importance, but the existence of an inner product is crucial for using the kernel function k instead of explicit evaluation of inner product in the Hilbert space. Because of this theorem, positive definite kernel functions are also called Mercer kernels.

Linear Kernel

When dealing with data that is linearly separable in the original input space, there's no necessity to project the input into a high-dimensional space. In such scenarios, a linear kernel can be employed, which essentially represents the dot product of two vectors within the original space:

$$k(x, y) = \langle x, y \rangle$$

Polynomial Kernel

Polynomial kernels of degree p are given by:

$$k(x, y) = (\langle x, y \rangle + 1)^p$$

Radial Basis Function Kernel

$$k(x, y) = \exp\left(-\frac{\|x-y\|^2}{2\sigma^2}\right)$$

where σ is the parameter which controls the radius and $\sigma > 0$.

Sigmoid Kernel

$$k(x, y) = \tanh(m\langle x, y \rangle + c)$$

,for some (not every) m > 0 and c > 0

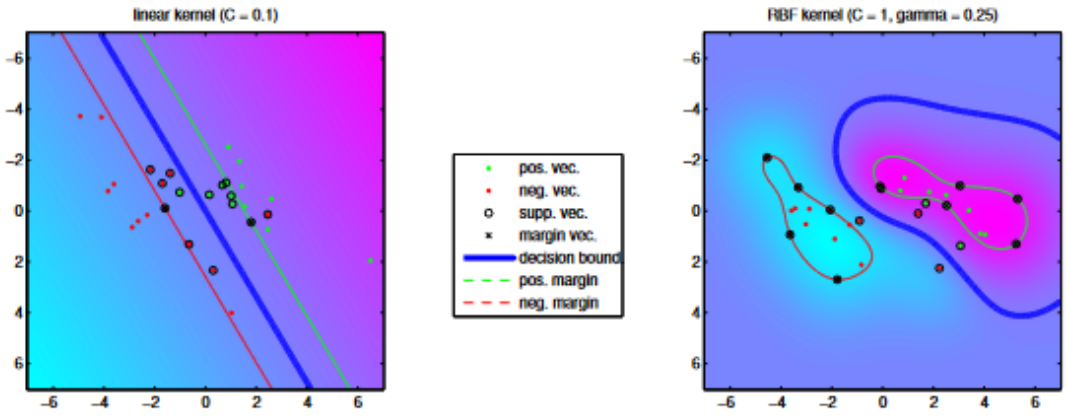


Figure 2.2: Boundaries obtained using SVMs with linear and RBF kernels [5]

2.2 Pattern Classification with SVM

Creating the process of searching for the optimal hyperplane for a multi-class classification problem, we are in a position to optimize formally and systematically our support vector machine for a work of classifying patterns [1].

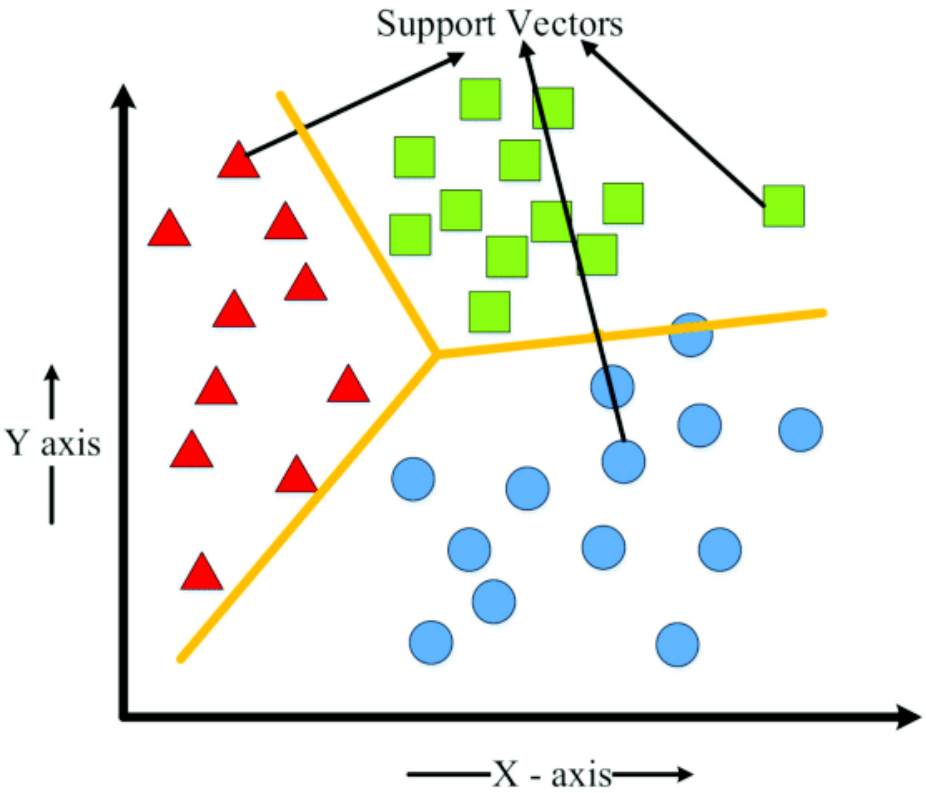


Figure 2.3: Multi-class Classification

Specifically, the way of ”thinking” of support vector machine is divided into two mathematical processes which are shown below.

1. The non-linear classificatino of an input vector into a feature space of higher dimensions, whom remains hidden for the input and the output.

2. The build of an optimal hyperplane for the distinction of the features as discovered before.

The number of features that are part of the hidden layer is derived from the number of support vectors. So, the SVM theory presents an analytical algorithm for the catego-

rization of the hidden (inner) layer of features, securing this way the optimality of the classification process.

Chapter 3

Methodology

3.1 Dataset Description

The dataset [2] covers a wide range of fruits, contributing this way to the development of machine learning models for intelligent software applications.

3.1.1 Classes Description

The dataset consists of 14700+ high-quality images of 6 different classes of fruits in the processed format. The images are divided into 3 sub-folders

1. Good quality fruits
2. Bad quality fruits
3. Mixed quality fruits.

Each sub-folder contains the 6 fruits images: apple, banana, guava, lime, orange, and pomegranate in .jpg format.

Dataset Statistics

In this section we provide some general information about the dataset with regard to its size for each of the 3 sub-folders. The total size of the dataset is 3.29 GB. In Table 1.1 you can see the total size of the sub-folders and in Table 1.2 the size and number of images per class.

Folder	Size
Bad Quality_Fruits	131 MB
Good Quality_Fruits	1.87 GB
Mixed Quality_Fruits	13.5 MB

Table 3.1: Fruit Quality Sub-folders

Class	Number of Images	Size	Format
Apple_Bad	1141	20.7 MB	.jpg
Banana_Bad	1087	24.6 MB	.jpg
Guava_Bad	1129	22.1 MB	.jpg
Lime_Bad	1085	19.3 MB	.jpg
Orange_Bad	1159	20.8 MB	.jpg
Pomegranate_Bad	1187	23.5 MB	.jpg
Apple_Good	1149	17.4 MB	.jpg
Banana_Good	1113	29.5 MB	.jpg
Guava_Good	1152	20 MB	.jpg
Lime_Good	1094	18.3 MB	.jpg
Orange_Good	1216	25.8 MB	.jpg
Pomegranate_Good	5940	1.76 GB	.jpg
Apple	113	2.09 MB	.jpg
Banana	285	3.78 MB	.jpg
Guava	148	1.53 MB	.jpg
Lemon	278	2.61 MB	.jpg
Orange	125	1.62 MB	.jpg
Pomegranate	125	1.92 MB	.jpg

Table 3.2: Training Fruit Quality and Variety Dataset

3.1.2 Testing Dataset Description

To ensure an unbiased dataset, a test set was created using images sourced from Google. The specifics of this dataset, including the statistics and sizes, are detailed in Table 1.3. But we keep in mind that the two datasets were not created under the same circumstances.

Class	Number of Images	Format
Apple_Bad	10	.jpg
Banana_Bad	10	.jpg
Guava_Bad	10	.jpg
Lime_Bad	10	.jpg
Orange_Bad	10	.jpg
Pomegranate_Bad	10	.jpg
Apple_Good	10	.jpg
Banana_Good	10	.jpg
Guava_Good	10	.jpg
Lime_Good	10	.jpg
Orange_Good	10	.jpg
Pomegranate_Good	10	.jpg
Apple	10	.jpg
Banana	10	.jpg
Guava	10	.jpg
Lemon	10	.jpg
Orange	10	.jpg
Pomegranate	10	.jpg

Table 3.3: Testing Fruit Quality and Variety Dataset

3.2 Data preprocessing

Since we are focusing on the fruit quality, a reduction in background was made to the images that initially included unnecessary information. A crucial initial step involves standardizing the size of all images to ensure uniformity across the dataset, which is vital for consistent analysis and comparison. So each image is resized to dimensions of 256x192 pixels. By this we ensure that all images are processed under a uniform scale, facilitating a more consistent feature extraction and analysis. Also, it optimizes computational efficiency, as working with images of the same size reduces variability in processing time and resource allocation.

3.2.1 Background reduction

The process beginds by defining a region of interest within each image. Given the dimensions of an image (width and height), we calculate an optimized rectangle that serves as an initial guess of the fruit's location. This rectangle is centrally placed and covers 80% of the image area—calculated by reducing the image's width and height by 20% (10% from each side).

GrabCut algorithm

With the region of interest defined, the GrabCut algorithm[3] is applied to segment the foreground (fruit) from the background. GrabCut uses a combination of machine learning techniques and graph theory to model the image's elements, distinguishing between the foreground and background based on color statistics and pixel distribution within the predefined rectangle. The algorithm iteratively refines the segmentation by estimating the color distributions of the foreground and background using Gaussian Mixture Models (GMMs). It treats image segmentation as a graph-cut problem, where the graph nodes represent pixels, and the edges define the similarity between neighboring pixels.

After applying the GrabCut algorithm, a binary mask is generated, indicating the foreground and background pixels. This mask is then refined to ensure a clear separation, with the foreground pixels set to 1 and the background pixels set to 0. Multiplying the original image by this mask isolates the foreground (fruit) by making the background transparent (or black, depending on the final representation needs).

This preprocessing step, starting with image resizing followed by background reduction, is crucial for enhancing the focus on the fruit, thereby reducing the classification algorithm's susceptibility to variations in the background. It ensures that the classifier can more accurately analyze the fruit's features, leading probably to improved classification accuracy.

3.3 Feature Extraction

In our problem of fruit quality classification, we need to identify and quantify characteristics of fruit through image analysis. The feature extraction stage is designed to distill critical information from fruit images that are indicative of quality, such as color, texture, and shape. This process transforms raw data into a format that is more manageable and informative for machine learning models. We employ a multi-faceted approach to feature extraction, utilizing various computational techniques to capture a comprehensive set of features.

Features

1. **Color Features (HSV Color Space):** The extract_color_histogram_hsv function focuses on extracting color features by converting images from the RGB to the HSV (Hue, Saturation, Value) color space. Color histograms for the hue and saturation channels are computed and normalized to form a feature vector. The HSV space is particularly chosen for its effectiveness in capturing color information that more closely aligns with human color perception, making it highly relevant for assessing the visual quality of fruits.

2. **Texture Features (Local Binary Pattern - LBP):** Texture is a critical indicator of fruit quality, reflecting surface conditions and maturity. The extract_lbp_features function applies the Local Binary Pattern (LBP) method to extract this information. By converting images to grayscale and computing the LBP, we obtain a histogram of LBP values that is normalized and used as a texture feature vector. This method effectively captures micro-patterns and textures, which are essential for distinguishing between different quality levels of fruit.

3. **Shape Features (Contours):** The shape of a fruit can reveal a lot about its quality, including the presence of deformities. Through the extract_shape_features function, shape-based features are derived by identifying contours in the grayscale image. A simple informative shape feature is the number of contours, which can indicate variations in fruit shapes that are characteristic of their quality.

Our method of classifying fruit quality is based mostly on the extraction of attributes related to color, texture, and shape. We provide the SVM classifier with a comprehensive set of data points that accurately represent the complex nature of fruit quality by carefully examining these factors. This approach not only increases the classification model's accuracy but also presents the precise traits that have the most effects on fruit quality. We ensure that our model is well-positioned to produce knowledgeable and trustworthy classifications through this thorough feature extraction method, opening the door for automated quality assessment in the agriculture industry.

3.4 SVM model

For our task we are using Support Vector Machine (SVM) models. We choose SVMs for their effectiveness in high-dimensional spaces and their ability to handle non-linear data through the use of kernel functions. We specifically test four variations of SVMs kernels: Linear, Radial Basis Function (RBF), Polynomial, Sigmoid, to ascertain their performance across our multi-class classification problem.

3.5 Model Training and Validation

To prepare our dataset for the SVM model training and evaluation, we employed a stratified sampling approach to split the preprocessed data into training and testing sets. This method ensures that both sets reflect the overall composition of the original dataset concerning the fruit quality labels. Specifically, we divided the data within each class label into training (80%) and testing (20%) subsets, maintaining the proportion of each class label across both datasets. This approach aimed to achieve a balanced representation of classes, critical for the unbiased training and fair evaluation of our models. After splitting, we randomized the order of instances in both the training and testing datasets to prevent any potential bias that could arise from the original data ordering. This step is crucial for ensuring that the models' learning is generalized and not influenced by the sequence of data.

Upon reviewing the class distribution (Table 3.5) in the training and testing sets, we observed a significant imbalance among classes. Class imbalance poses a challenge for machine learning models, as it can bias the model towards the majority class and result in poor predictive performance on minority classes.

Labels	**Count**
Pomegranate_Good	3801
Orange_Good	972
Pomegranate_Bad	949
Orange_Bad	927
Guava_Good	921
Apple_Bad	912
Apple_Good	907
Guava_Bad	903
Banana_Good	890
Lime_Good	875
Banana_Bad	869
Lime_Bad	868
Banana_Mixed	228
Lemon_Mixed	222
Guava_Mixed	118
Orange_Mixed	100
Pomegranate_Mixed	100
Apple_Mixed	90

Table 3.4: Training Set Class Distribution

To solve this issue in the **training set**, we follow two approaches: trimming and

augmentation. Trimming involved reducing the number of instances in the overrepresented classes to better match those in the underrepresented ones, preventing the model from being overwhelmed by the majority classes. For the data augmentation we transform the images using the following trainsformations:

1. 90°rotation
2. 180°rotation
3. 270°rotation
4. Gamma 0.2
5. Gamma 1.2

Hyperparameter tuning SVM

A comprehensive hyperparameter tuning exercise using the SVMs kernels mentioned in section 3.4 was performed out. Our parameter grid was designed to explore a wide range of configurations, focusing on the regularization parameter C, the kernel type, and the gamma parameter for kernels where it is applicable. The C parameter, crucial for trading off correct classification of training examples against maximization of the decision function's margin, was tested across values [0.1, 0.5, 1, 10] to find the optimal balance between model complexity and generalization capability. For kernels requiring the gamma parameter, which determines the influence of individual training examples, we explored a variety including 'scale', 'auto', and explicit values [0.1, 0.01, 0.001, 0.0001], aiming to fine-tune the models' sensitivity to feature space. To address potential class imbalance issues, we also varied the class_weight parameter between None and 'balanced', the latter automatically adjusting weights inversely proportional to class frequencies in the training data.

Our grid search was executed with a 5-fold cross-validation strategy. This approach allowed us to systematically evaluate and compare the performance of different SVM configurations, optimizing not just for accuracy but also considering the balance between precision and recall, especially critical in a multi-class classification context. The selection of the best model was based on its cross-validated performance (f1 macro), leading to the identification of an optimal set of hyperparameters that promised the most effective balance between prediction accuracy and model generalizability across our diverse dataset.

3.6 Results analysis

After the completion of the grid search the best SVM (Figure 3.6) is:

1. kernel: rbf
2. C: 10
3. class_weight: balanced
4. gamma: scale

The confusion matrix for our classifier, presented visually, highlights the true positive predictions along the diagonal, where the predicted labels match the true labels (Table 3.6). Classes with the darkest shades along the diagonal, such as class 0, 3, and 16, represent the highest number of correct predictions, indicating that our model has learned to identify these classes with high reliability. However, certain classes displayed lighter shades on the diagonal and darker off-diagonal elements, revealing a trend where the model has confused these classes with others. For instance, class 1 has a noticeable number (35%) of observations that were predicted to be in other classes, suggesting that the model may struggle to distinguish these particular classes accurately. Furthermore, the overall metrics—such as a weighted precision of 0.90, a recall of 0.89, and an F1 score of 0.89—indicate a robust model that performs well across the majority of classes.

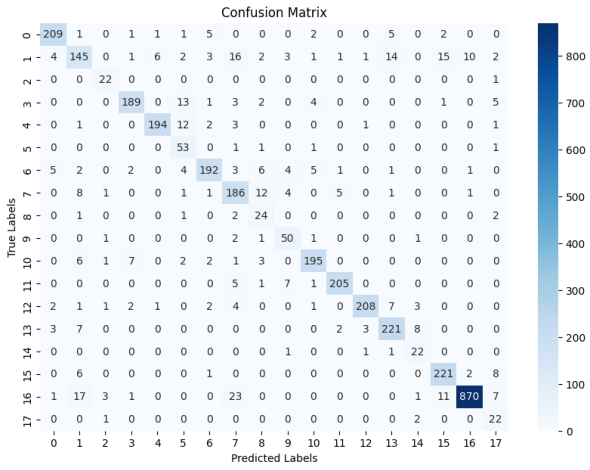
Encoded Value	Original Label
0	Apple_Bad
1	Apple_Good
2	Apple_Mixed
3	Banana_Bad
4	Banana_Good
5	Banana_Mixed
6	Guava_Bad
7	Guava_Good
8	Guava_Mixed
9	Lemon_Mixed
10	Lime_Bad
11	Lime_Good
12	Orange_Bad
13	Orange_Good
14	Orange_Mixed
15	Pomegranate_Bad
16	Pomegranate_Good
17	Pomegranate_Mixed

Table 3.5: Encoded Values and Original Labels

Figure 3.1: Best SVM kernel, parameters

```
Fitting 5 folds for each of 192 candidates, totalling 960 fits
Best hyperparameters: {'C': 10, 'class_weight': 'balanced', 'gamma': 'scale', 'kernel': 'rbf'}
Accuracy: 0.89
Precision (Weighted): 0.90
Recall (Weighted): 0.89
F1 Score (Weighted): 0.89
              precision    recall  f1-score   support

           0       0.93      0.92      0.93       227
           1       0.74      0.64      0.69       226
           2       0.73      0.96      0.83        23
           3       0.93      0.87      0.90       218
           4       0.96      0.91      0.93       214
           5       0.60      0.93      0.73        57
           6       0.92      0.85      0.88       226
           7       0.75      0.85      0.79       220
           8       0.46      0.80      0.59        30
           9       0.72      0.89      0.80        56
          10       0.92      0.90      0.91       217
          11       0.96      0.94      0.95       219
          12       0.97      0.90      0.93       232
          13       0.88      0.91      0.89       244
          14       0.59      0.88      0.71        25
          15       0.88      0.93      0.91       238
          16       0.98      0.93      0.96       934
...
    accuracy                           0.89      3631
   macro avg       0.80      0.88      0.83      3631
weighted avg       0.90      0.89      0.89      3631
```



Our SVM classifier was trained on a dataset where the images largely share a uniform background, which, while advantageous for initial model training and internal validations, may have inadvertently introduced a bias towards this specific feature. So

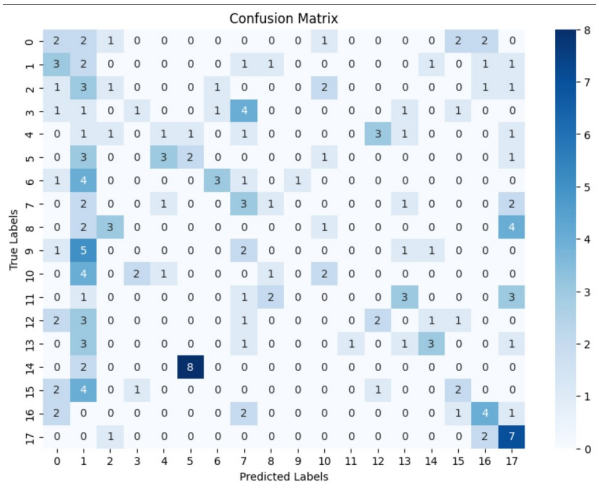
we created a new custom test set to evaluate our classifier and we got the following results (Figure 3.6):

Figure 3.2: Custom test set evaluation

```
Accuracy: 0.18
Precision (Weighted): 0.19
Recall (Weighted): 0.18
F1 Score (Weighted): 0.17
              precision    recall  f1-score   support

           0       0.13      0.20      0.16        10
           1       0.05      0.20      0.08        10
           2       0.14      0.10      0.12        10
           3       0.25      0.10      0.14        10
           4       0.17      0.10      0.12        10
           5       0.18      0.20      0.19        10
           6       0.60      0.30      0.40        10
           7       0.18      0.30      0.22        10
           8       0.00      0.00      0.00        10
           9       0.00      0.00      0.00        10
          10       0.29      0.20      0.24        10
          11       0.00      0.00      0.00        10
          12       0.33      0.20      0.25        10
          13       0.12      0.10      0.11        10
          14       0.00      0.00      0.00        10
          15       0.29      0.20      0.24        10
          16       0.40      0.40      0.40        10
          17       0.32      0.70      0.44        10

    accuracy                           0.18       180
   macro avg       0.19      0.18      0.17       180
weighted avg       0.19      0.18      0.17       180
```



As we can see the classifier is not performing well on this new data, with an overall accuracy of 0.18, which is significantly lower than what might be expected based on the initial training and validation. First of all, this is probably happening because the custom test set differ significantly from the training data in various aspects, such as image quality, angles, lighting conditions, and background complexity. Secondly, the size and composition of the custom test set also pose significant challenges. With only 10 images per class, the sample is arguably too small to capture the broader, real-world variability of the data. This limitation could result in high variance in the model's performance metrics, rendering them an unreliable indicator of true model capability. Also we need to keep in mind that the training set includes Indian fruits and the custom set does not include Indian fruits. In light of these observations, it is clear that enhancing the model's generalization capabilities is necessary. This can be addressed by expanding the diversity of the training data to include a wider range of fruit varieties and backgrounds, and by implementing domain adaptation techniques.

3.6.1 Comparison with other models

KNN

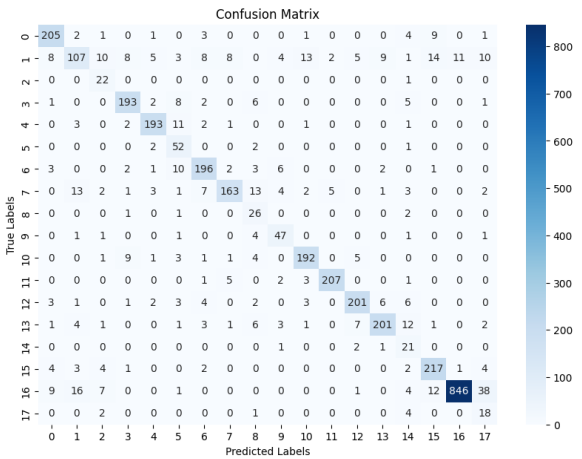
This section presents the classification report and confusion matrix for a KNN model. The best hyperparameters identified for the model are a Manhattan distance metric and a count of 3 neighbors. The model achieves an accuracy of 0.85. Other key metrics include a weighted average precision of 0.88, recall of 0.85, and an F1 score of 0.86. The classification report table lists precision, recall, f1-score, and support for each class labeled from 0 to 17, indicating a multi-class classification problem with 18 classes. The support column represents the number of true occurrences of each class in the dataset.

The confusion matrix on the right visualizes the performance of the model, with predicted labels on the x-axis and true labels on the y-axis. The color gradient represents the number of predictions, with darker colors indicating higher numbers. The model seems to perform well for most classes with some confusion noticeable between certain classes, as indicated by the non-zero values off the diagonal.

```
Best hyperparameters: {'metric': 'manhattan', 'n_neighbors': 3}
Accuracy: 0.86
Precision (Weighted): 0.88
Recall (Weighted): 0.86
F1 Score (Weighted): 0.86
              precision    recall  f1-score   support

           0       0.88      0.90      0.89       227
           1       0.71      0.47      0.57       226
           2       0.43      0.96      0.59        23
           3       0.89      0.89      0.89       218
           4       0.92      0.90      0.91       214
           5       0.55      0.91      0.68        57
           6       0.86      0.87      0.86       226
           7       0.90      0.74      0.81       220
           8       0.39      0.87      0.54        30
           9       0.70      0.84      0.76        56
          10       0.89      0.88      0.89       217
          11       0.97      0.95      0.96       219
          12       0.91      0.87      0.89       232
          13       0.91      0.82      0.87       244
          14       0.30      0.84      0.45        25
          15       0.85      0.91      0.88       238
          16       0.99      0.91      0.94       934
          17       0.23      0.72      0.35        25

    accuracy                           0.86      3631
   macro avg       0.74      0.85      0.76      3631
weighted avg       0.88      0.86      0.86      3631
```



Decision Tree

This images display the classification report and confusion matrix for a Decision Trees model. The model's accuracy is 0.63. The weighted averages for precision, recall, and F1 score are 0.63, 0.63, and 0.62, respectively, suggesting a relatively balanced performance across classes, but with room for improvement.

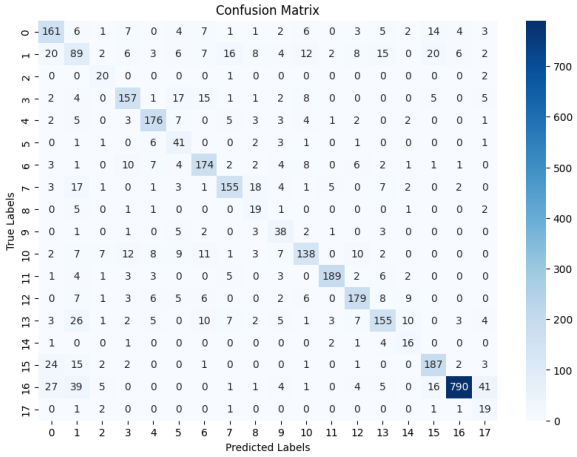
The confusion matrix on the right visualizes the performance of the model, with predicted labels on the x-axis and true labels on the y-axis. The color gradient represents the number of predictions, with darker colors indicating higher numbers.. In this

matrix, while there are many correct predictions, there are also notable amounts of misclassifications , especially for certain classes. This suggests that the model has specific difficulties or confusions between some classes.

```
Fitting 5 folds for each of 36 candidates, totalling 180 fits
Best hyperparameters: {'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 5}
Accuracy: 0.74
Precision (Weighted): 0.78
Recall (Weighted): 0.74
F1 Score (Weighted): 0.76
              precision    recall  f1-score   support

           0       0.65      0.71      0.68       227
           1       0.39      0.39      0.39       226
           2       0.45      0.87      0.60        23
           3       0.75      0.72      0.74       218
           4       0.81      0.82      0.82       214
           5       0.41      0.72      0.52        57
           6       0.74      0.77      0.76       226
           7       0.79      0.70      0.75       220
           8       0.30      0.63      0.41        30
           9       0.46      0.68      0.55        56
          10       0.73      0.64      0.68       217
          11       0.93      0.86      0.90       219
          12       0.80      0.77      0.79       232
          13       0.73      0.64      0.68       244
          14       0.36      0.64      0.46        25
          15       0.77      0.79      0.78       238
          16       0.98      0.85      0.91       934
          17       0.23      0.76      0.35        25

    accuracy                           0.74      3631
   macro avg       0.63      0.72      0.65      3631
weighted avg       0.78      0.74      0.76      3631
```



Chapter 4

System Integration

4.1 Implementation

In this section we have the implementation, which is a live demo of our project. We assembled a Raspberry Pi system that is tacking pictures and then process to the classification via our server.

4.1.1 Raspberry Pi system

Hardware Setup

We assembled our prototype using a selected array of hardware components, as illustrated in Figure 4.5

- Raspberry Pi Model: Raspberry Pi 3 Model B
- Camera Module: Raspberry Pi Camera Board v1.3 (5MP, 1080p)
- Button

In our prototype, the camera module is connected with the Raspberry Pi 3 by connecting it to the dedicated CSI (Camera Serial Interface) port. This connection is facilitated by a flexible ribbon cable that allows a stable communication between the camera and the main board. Moreover, the system includes a button which acts as the trigger for initiating the image capture. It is set with a pull-up resistor, which means it will read high (1) when not pressed and low (0) when pressed. This button is connected to the general-purpose input/output (GPIO) pin 16 on the Raspberry Pi.

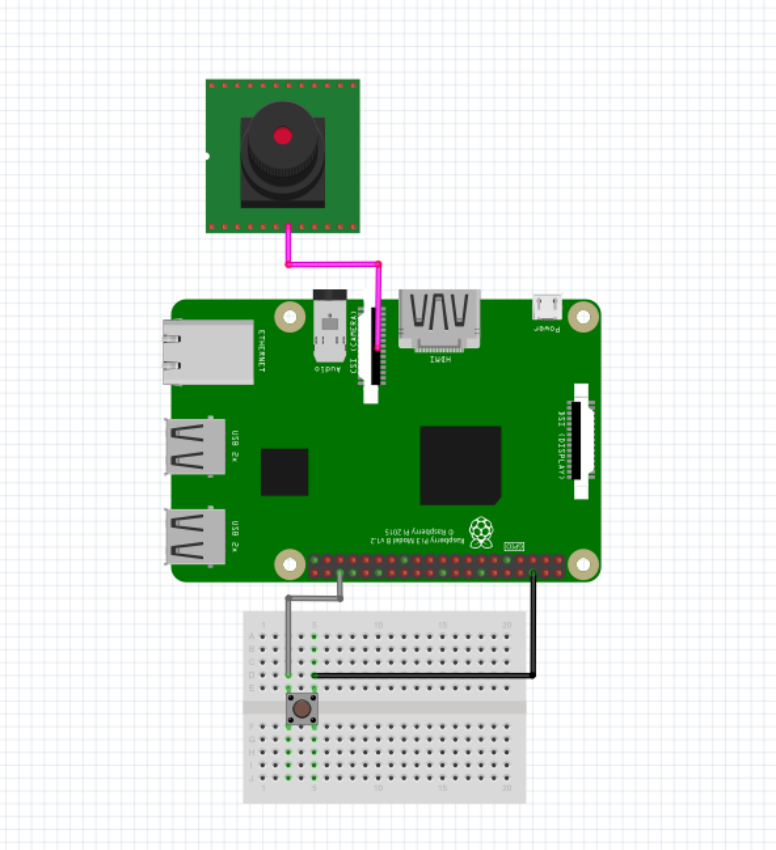


Figure 4.1: Raspberry Pi circuit

Software Configuration

- Operating System: Raspbian GNU/Linux
- Internet connection

The code 4.1 captures an image and send it to our server. Firstly, it sets up the Raspberry Pi's GPIO pin 16, to listen for a button press. After, the script enters an infinite loop, continuously checking the state of the button. When the button is detected as pressed, the Raspberry Pi triggers the camera to start a preview, capture an image, and save it to the specified location on the Raspberry Pi's filesystem. After capturing the image, the code makes an HTTP POST request to send the image to our server. The format of the data sent is form-data, which is standard for file uploads. Finally, the code is enclosed in a try-except block to ensure that the GPIO pins are properly cleaned up if the program is interrupted, preventing the pins from being left in an uncertain state.

```
1
```

```
import RPi.GPIO as GPIO
import time
from picamera import PiCamera
from time import sleep
import requests

camera = PiCamera()
url = 'http://94.65.90.245:6060/upload'

BUTTON_PIN = 16
GPIO.setmode(GPIO.BCM)
GPIO.setup(BUTTON_PIN, GPIO.IN, pull_up_down=GPIO.PUD_UP)

try:
    while True:
        time.sleep(0.1)
        if GPIO.input(BUTTON_PIN) == GPIO.LOW:
            camera.start_preview(alpha=192)
            camera.capture("/home/pi/Desktop/buttonimage.jpg")
            camera.stop_preview()
            files = {'image': open('/home/pi/Desktop/buttonimage.jpg', 'rb')}
            response = requests.post(url, files=files)
            print(response.text)
except KeyboardInterrupt:
    GPIO.cleanup()
```

Listing 4.1: Code for triggering Raspberry Pi camera with a button and sending the image to a server.

Hardware implementation

4.1.2 Detailed steps of implementing the server/ui system

The server is constructed using Flask, a micro web framework for Python. It functions as an image upload platform with supplementary capabilities like data processing and image labeling using the trained SVM model. The server offers an endpoint that manages POST requests for image uploads. The received images are stored in a local folder, and the server responds with a success message indicating that the image has been received and stored. Additionally, there is an endpoint that showcases the images on a webpage along with their respective labels.

4.1.3 Screenshots - Webpage

The webpage features a user-friendly interface, where each fruit is clearly labeled along with its assessed quality. For each fruit, if a photo has been successfully captured and

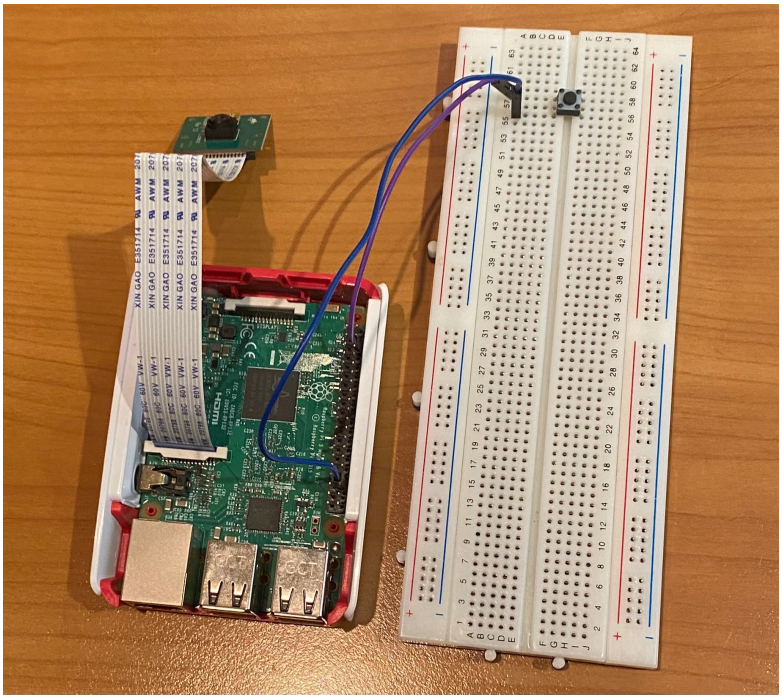


Figure 4.2: Raspberry Pi implementation

classified, that photo is prominently displayed next to the fruit's label (Figure 4.1.3). In cases where no photo is available due to classification or capture issues, a placeholder image is shown instead 4.1.3.

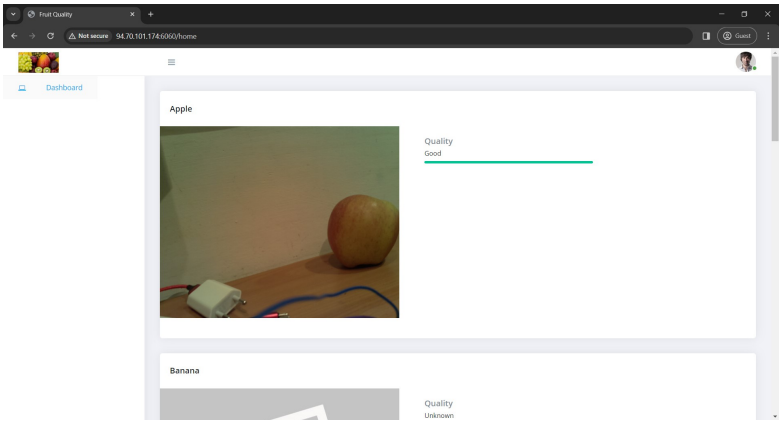


Figure 4.3: Website example 1

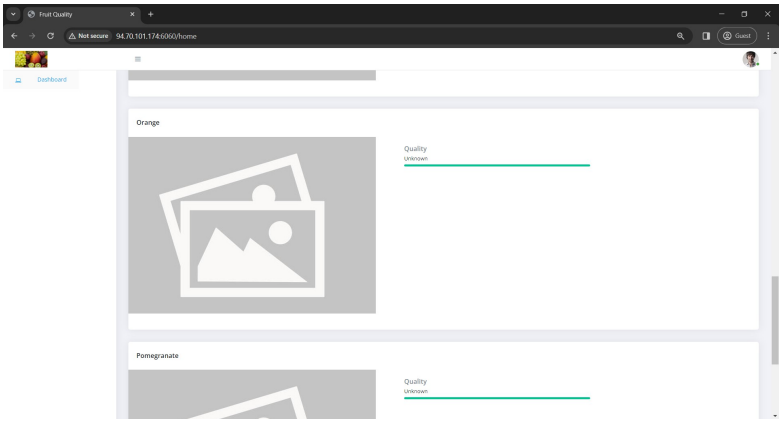


Figure 4.4: Website example 2

4.1.4 Use case

The following simple diagram (Figure 4.1.4 shows the sequence of operations beginning with the Raspberry Pi 3B capturing an image of the fruit, which is then sent to the server through a POST request. The server, housing the model and images, processes the image to determine the fruit's quality. Lastly, the processed data is returned to the user through the site interface.

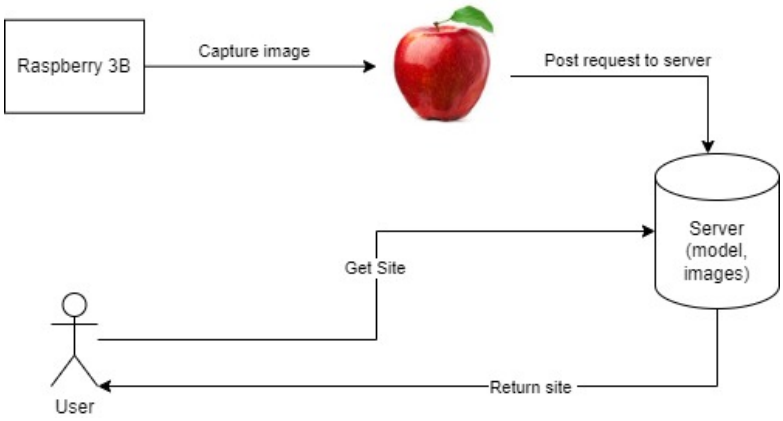


Figure 4.5: Website example 2

Chapter 5

Discussion & Future Work

In image analysis tasks, CNNs have proven to be remarkably capable, especially when it comes to capturing complex features and spatial relationships within images. The classification pipeline may be able to assess fruit quality more thoroughly and accurately if CNNs are integrated. The system may be able to more effectively adjust to the complexities and variations found in fruit images by utilizing CNN architectures created especially for image classification tasks. This could result in higher classification accuracies.

Furthermore, by utilizing the knowledge gained from extensive image datasets, investigating methods like transfer learning with pre-trained CNN models could speed up the development process. Thus, further research into CNNs' applicability for classifying fruit quality and how best to incorporate them into the current framework may be beneficial [4].

Bibliography

[1] S.S. Haykin. *Neural Networks and Learning Machines*. Pearson International Edition. Pearson, 2009. ISBN: 9780131293762. URL: https://books.google.gr/books?id=KCwWOAAACAAJ.

[2] Shashwat Kumar. *FruitNet: Indian Fruits Dataset with Quality*. https://www.kaggle.com/datasets/shashwatwork/fruitnet-indian-fruits-dataset-with-quality?resource=download. Accessed: February 19, 2024. 2022.

[3] OpenCV Contributors. *OpenCV Documentation: GrabCut Algorithm*. https://docs.opencv.org/3.4/d8/d83/tutorial_py_grabcut.html. Accessed: February 19, 2024. 2022.

[4] Neha Sharma, Vibhor Jain, and Anju Mishra. "An Analysis Of Convolutional Neural Networks For Image Classification". In: *Procedia Computer Science* 132 (2018). International Conference on Computational Intelligence and Data Science, pp. 377–384. ISSN: 1877-0509. DOI: https://doi.org/10.1016/j.procs.2018.05.198. URL: https://www.sciencedirect.com/science/article/pii/S1877050918309335.

[5] Sreekanth Vempati. "Efficient SVM based Object Classification and Detection". Hyderabad, India: International Institute of Information Technology, Dec. 2010.

[6] Ioannis Vlahavas et al. *Artificial Intelligence*. 3rd ed. Έκδοση/Διάθεση: Εκδόσεις Πανεπιστημίου Μακεδονίας. Thessaloniki, Greece: University of Macedonia Press, 2011. ISBN: 978-960-8396-64-7.