Instant Identification of Bacteria Species using Integration of Colorimetric Sensing Arrays And Deep Learning

(Mostafa – Akbari – Karan, MAK - 1)

By

Karanvir Singh

Bachelor of Technology, Punjab Engineering College, 2022

A Project Submitted in Partial Fulfillment of
the Requirements for the Degree of

MASTER OF ENGINEERING

In the Department of Mechanical Engineering
University of Victoria, Victoria, BC,
Canada

Instant Identification of Bacteria Species using Integration of Colorimetric Sensing Arrays And Deep Learning

(Mostafa – Akbari – Karan, MAK 1)

By

Karanvir Singh

Bachelor of Technology, Punjab Engineering College, 2022

Supervisory Committee:

Faculty Supervisor: Dr. Mohsen Akbari

External Examiner: Dr. Somayeh Fardindoost

ABSTRACT

The research delves into the integration of colorimetric sensors in detecting volatile organic compounds (VOCs) for rapid bacterial identification through advanced machine-learning algorithms. With the use of a colorimetric sensor array that detects any VOCs in the form of a chemical change, we were able to establish a methodology. The pattern was formed and further deep analysis of this pattern to produce homogeneity in results was the goal. This method uses optoelectronic arrays to process RGB data, allowing for highly specific bacterial sample separation. Artificial intelligence frameworks are used in the creation and testing to improve detection capabilities and increase accuracy even with little data. The final ANN model utilized for the image classification was able to produce 92% accuracy within 2 minutes after utilizing a training sample of 235 samples and testing it on 10% of data throughout the span of 2 months. The results of the findings extend to clinical diagnostics, where accurate detection might facilitate targeted treatments and expedite pathogen identification. The results indicate potential for practical application, providing a robust tool for non-invasive bacterial classification.

Keywords: Colorimetric Sensor Array (CSA), Volatile Organic Compounds (VOCs), Bacterial Identification, Deep Learning, Artificial Neural Network (ANN), Machine Learning, Wound Diagnostics, Image Analysis

CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGEMENT

First and foremost, I want to sincerely thank my supervisor, Dr. Mohsen Akbari, for giving me the chance to attend the University of Victoria and for his guidance and assistance during my master's program. His knowledge and direction have greatly influenced this study, and I am appreciative of his thoughtful counsel regarding my research question.

Additionally, I would like to express my gratitude to Dr. Mostafa Azimzadeh, a post-doctoral researcher at the University of Victoria, for his kind help in determining the project's layout and for his insightful appraisal of my work. He introduced me to the concept and showed me the scope to which we can expand this idea and develop a useful prototype. I couldn't have learned as much as I did here without them. I am grateful for their amazing assistance and advice.

I want to thank all my other teachers, coworkers, friends, and siblings for their support during the journey at the University of Victoria. I am grateful as I feel that this experience will change my life further.

In addition, I would like to express my gratitude to the University of Victoria for granting me permission to work and study for my master's degree in mechanical engineering, as well as for providing the tools and a supportive environment for this project and my program.

Finally, I want to express my gratitude to my parents for their unwavering love, encouragement, and support along this journey. Unquestionably, reaching this milestone would be difficult without them.

We acknowledge and respect the lək̓ʷəŋən peoples on whose traditional territory the university stands, and the Songhees, Esquimalt and W̱SÁNEĆ peoples whose historical relationships with the land continue to this day.

I. INTRODUCTION

The timely identification of bacterial species is a crucial factor in both medical diagnostics and industrial applications. Traditional methods, such as culturing techniques and biochemical assays, often require significant time and resources, delaying potential treatment and increasing the risk of severe health outcomes, such as sepsis and other life-threatening conditions [1]. The advent of electronic and optoelectronic sensing technology has opened new avenues for rapid bacterial identification by detecting volatile organic compounds (VOCs) emitted by bacterial metabolism [2], [1].

Colorimetric sensor arrays often referred to as "artificial noses," have emerged as an effective tool for analyzing VOCs due to their sensitivity and ability to provide unique chemical "fingerprints" for different bacterial species [2], [1]. These sensors utilize chemo-responsive colorants that react with analytes, producing measurable changes in color that can be digitally captured and analyzed. This capability is particularly valuable for distinguishing between closely related bacterial strains, thereby facilitating precise identification and reducing the time required for diagnostic processes [2], [3].

Integrating deep learning with colorimetric sensing arrays enhances the analysis by automating the detection and interpretation of complex data. Advanced machine learning models, such as artificial neural networks (ANNs) and convolutional neural networks (CNNs), are employed to recognize patterns within high-dimensional data from sensor arrays [1]. These AI-driven techniques provide superior accuracy in classification tasks by learning intricate relationships within the data that might not be discernible through traditional analytical methods [1]. The integration of deep learning thus enables the robust processing of RGB data, advancing the efficacy of non-invasive bacterial detection and paving the way for scalable applications in clinical and industrial environments [3], [2].

This study explores the amalgamation of colorimetric sensor technology with deep learning

algorithms to develop a rapid, reliable, and cost-effective approach for the instant identification of bacterial species. By leveraging the chemical sensitivity of colorimetric arrays and the computational power of AI, this method seeks to address current limitations in diagnostic speed and accuracy [1], [2], [3].

1.1 Deep Dive into the Concept

Numerous sensing devices have been created to detect volatile organic compounds (VOCs), including e-noses, metal-oxide sensors, electrochemical detectors, surface-enhanced Raman spectroscopy (SERS), single-dye colorimetric sensors, and optoelectronic noses or colorimetric sensor arrays (CSA) featuring multiple color-changing dyes. Among these options, CSA, composed of multiple gas-sensitive dyes, provides a dependable and energy-independent method for identifying bacteria in particular applications, such as assessing meat spoilage, determining bacterial loads in water, detecting urinary tract and lung infections, and identifying potential bioterrorism agents. These instruments function by observing alterations in the optical characteristics of a dye array when exposed to specific VOCs. As VOC molecules interact with the dyes, they trigger color changes, resulting in a distinctive pattern of color variations corresponding to the type of bacteria on a given substrate. By examining the unique spectral patterns generated by different VOCs, CSA can precisely identify and quantify various bacterial species. The unique spectral patterns are detected through their RGB values to detect any change from the default RGB values. Their capacity to categorize bacterial species according to these patterns is further improved by the incorporation of cutting-edge machine learning and deep learning techniques [4]. According to studies, they are useful for quickly identifying bacteria, including E. coli and other pathogens in challenging conditions. Furthermore, CSAs are now dependable instruments for accurate bacterial identification and VOC quantification thanks to advancements in sensor design, such as standardizing dye arrays and improving imaging conditions [2], [7].

1.2 Our Application for Colorimetric Sensing Arrays

This research explores the potential of utilizing a colorimetric VOC sensor array (VOC-CSA) for wound bacteria identification, a novel application in this field. Some of the various applications of the VOCs in the medical industry are given in Figure 1.

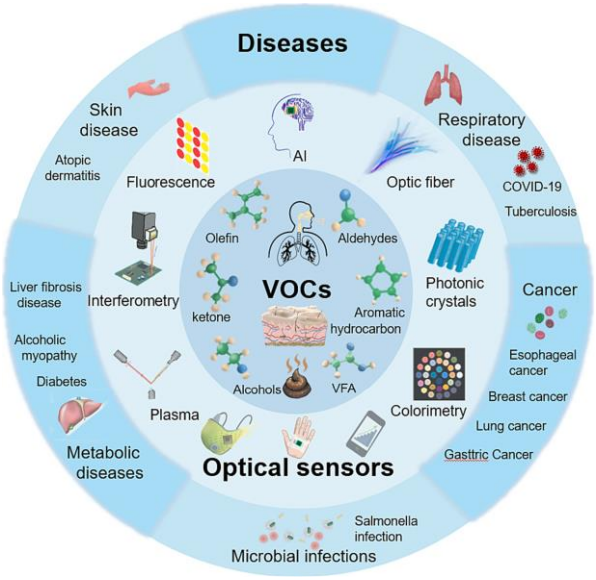


Figure 1. Various Applications of VOCs in the Medical Industry.

Infected wounds present a complex environment where bacterial interactions with the wound bed can obscure the specific VOC signature of the infecting organism. Relying exclusively on existing data, primarily derived from bacteria cultured on well-defined substrates such as agar, may compromise the accuracy and specificity of sensor-based detection. Moreover, infected wounds often contain multiple bacterial species, further complicating the VOC profile. In this study, we assessed the impact of substrate on VOC-CSA sensitivity and specificity using Luria-Bertani (LB) broth, a nutrient-rich medium commonly used for bacterial growth, agar, and ex vivo porcine ear skin grafts. Pig ears have been shown to possess the most relevant histology and wound-healing biochemistry in relation to human skin compared to other animal models. We focused on the three most frequently reported bacteria in human clinical wounds: *E. coli, P. aeruginosa, and S. aureus*. We examined both single-species and multi-species infections to evaluate the VOC-CSA's capability to distinguish between different bacterial species. The

research also aimed to determine the VOC-CSA's detection limit and specificity using advanced classification techniques. This analysis will contribute to assessing the practical applicability of this technology in clinical environments.

1.3 Fabrication of the Sensor Arrays

Our sensor design incorporated 30 dye spots attached to a polyvinylidene fluoride (PVDF) membrane. A diagram in Figure 2a from similar research works illustrates the sensor elements and various testing setups, including solid (agar) and liquid (broth) media, as well as an ex vivo study using porcine skin samples. These experiments involved photographing and examining the dye colors before and after exposure to bacterial VOC emissions. Dyes were chosen based on a literature review, followed by a methodical approach considering VOC reactivity and concentration optimization tests. Table 1 provides information on each dye's name, acidity or basicity, and its position on the PVDF membrane. Prior to application, the dye solutions were mixed thoroughly with a vortex mixer, and 1 l of each solution was precisely dispensed onto the membrane in a 6 x 6 array, as shown in Figure 2a. A 3D-printed mold was used to create circular indentations on the PVDF, forming enclosed circles with depressed edges to minimize lateral flow through the membrane and prevent dye mixing between areas.

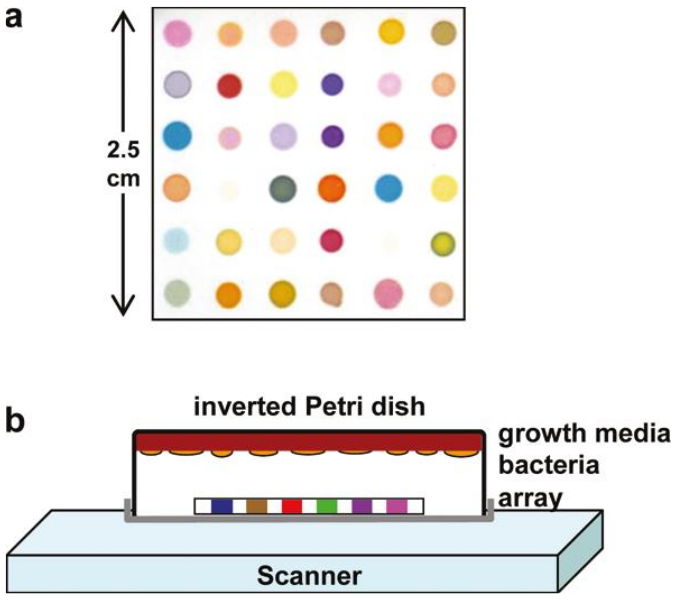


Figure 2. (a) Colorimetric Sensor Array Used for Bacterial Identification During Experiments; (b) Schematic of the Experimental Apparatus [3].

Table 1. 30 Dye Spots Description and Distribution on Colorimetric Sensing Array (CSA).

[A] – Acidic; [B] – Basic; [2] – Spot Number

Dye	Abbreviation & Spot Number
Brilliant yellow	**BY(A) [1]** **BY(B) [7]**
Phenol red	**PR(A) [4]** **PR(B) [10]**
Bromothymol blue	**BB(A) [2]** **BB(B) [27]**
Comassie blue	**CB [11]**
Richards Dye	**RI [28]** **RI(A) [5]**
Bromophenol red	**BPR(A) [6]** **BPR(B) [12]**
Naphthalene=alpha naphthol phthalein	**NP(A) [3]** **NP(B) [9]**
Porphine Cobalt (ll)	**COB [14]**
Porphine Copper (ll)	**COP [15]**
Porphine Zinc	**Zn [30]**
Porphine Zinc (ll)	**Zn2 [13]**
Hemin	**HE [26]**
Nile red	**NR [16]** **NR(A) [17]** **NR(B) [18]**
Bromocresol green	**BG [21]** **BG(A) [19]** **BG(B) [20]**
Cresol red	**CR [29]** **CR(A) [25]** **CR(B) [8]**
Nitrazine yellow	**NY [22]** **NY(A) [23]** **NY(B) [24]**

The reason for having 30 dye spots initially was because of their sensitivity to important metabolites generated during bacterial growth, these 30 dyes were chosen to detect the three bacterial species: *Staphylococcus aureus, Escherichia coli, and Pseudomonas aeruginosa*. These dyes were selected because they can detect a variety of volatile organic compounds (VOCs) and other metabolites, including amines, sulfides, and fatty acids. They include pH indicators (like Phenol Red, and Bromothymol Blue), porphyrins (like Porphine Cobalt, and sPorphine Zinc), and solvatochromic dyes (like Nile Red). For example, *P. aeruginosa* releases sulfur-containing

chemicals including pyocyanin, which can be detected by dyes based on phenol, while *E. coli creates* amines like putrescine and cadaverine, which interact with dyes like Bromophenol Red and Nitrazine Yellow. Because Cresol Red and Brilliant Yellow react well to pH changes, they are useful tools for monitoring *S. aureus*, which is known to produce fatty acids and staphylococcal toxins.

By offering a wider reaction range, the redundancy of some dyes (such as Brilliant Yellow and Nile Red with multiple spot designations) guarantees reliable detection. Furthermore, the incorporation of porphyrins, including Hemin and Porphine Zinc, takes advantage of their metal-binding capabilities to improve selectivity for particular VOC classes. High-dimensional information from this varied dye array is essential for distinguishing overlapping metabolic profiles in polymicrobial illnesses.

This sensor array is designed to precisely identify and categorize infections in a variety of clinical settings by focusing on the distinct metabolites that each bacterium produces. To encompass accurate predictions, we need a unique pattern for different bacteria, and choosing 30 spots initially was to see how all the changes can fit into data analytical techniques. Later, the dyes were narrowed to 9 spots based on various rationales specified in Section 9.3.

II. PROJECT OBJECTIVES AND OUTLINE

Title: Developing a Colorimetric Sensor Array Based on VOCs for Bacterial Identification in Complex Wound Settings using Deep Learning methods.

Objective: With an emphasis on wound-related bacteria like *Escherichia coli, Pseudomonas aeruginosa, and Staphylococcus aureus*, the goal is to create and assess a colorimetric VOC Colorimetric sensor array (VOC-CSA) that can identify and detect bacterial species based on their volatile organic compound (VOC) emission. The chemical change detected by the sensors will be processed by a neural network ANN or CNN that accepts imaging data and makes accurate predictions for the classification of bacterial infection. This research comprises the following phases and steps:

1. Reviewing the literature for bacteria detection methods like culture and molecular methods.

2. Examine the role of VOC analysis in medical diagnostics and its application in the non-invasive detection of bacterial infections. Understanding the VOC analysis in medical diagnostics and its usability in non-invasive detection of bacterial infections.

3. Developing a proper layout for developing the colorimetric sensors on a (PVDF) membrane.

4. Choose appropriate dyes based on literature for VOC reactivity and optimize concentrations. Selecting the appropriate dyes for our study based on their chemical changes and reactions to different bacterial samples.

5. Prepare bacterial cultures using Luria-Bertani (LB) broth and agar and ex vivo porcine skin to simulate wound environments.

6. Hands-on experimentation in the lab to test the developed sensors and record the VOC emission from single and mixed bacterial cultures.

7. Capture color change data with standardized imaging and process it to extract RGB

values for analysis.

8. Implement statistical and machine learning techniques, such as PCA and HCA, for multivariate analysis of collected data. Finding appropriate statistical unsupervised machine learning techniques for classification based on standard clustering approach.

9. Evaluate the limitations of the sensor and limitations in the machine learning techniques.

10. Compare outcomes across different substrates (agar, broth, and ex vivo skin) to validate sensor performance.

11. Address any observed limitations in sensor response time or environmental sensitivity.

12. Refine dye selection and sensor array design for enhanced detection accuracy.

III. VOLATILE ORGANIC COMPOUNDS (GENERATED BY SENSORS)

Volatile Organic Compounds (VOCs) are unique chemical emissions produced during bacterial metabolic processes, serving as indicators for species identification. VOCs include amines, sulfides, fatty acids, and other metabolites that vary by species and growth conditions. These compounds are formed through enzymatic reactions, such as the breakdown of amino acids or lipids, and they play a crucial role in bacterial interaction and survival [4].

Colorimetric sensor arrays (CSAs) utilize dyes that react with VOCs by changing color, driven by specific chemical interactions like acid-base reactions, redox processes, and solvatochromic effects. When VOCs contact the array, they interact with chemically responsive dyes, altering their optical properties. This change is detected as a shift in the RGB values, forming unique color patterns that correspond to the VOC profile of a specific bacterial species [2][3].

The use of CSAs allows for rapid, non-invasive detection and differentiation of bacteria through the analysis of these color changes, which can be processed using statistical methods to accurately classify and identify different bacterial species [5].

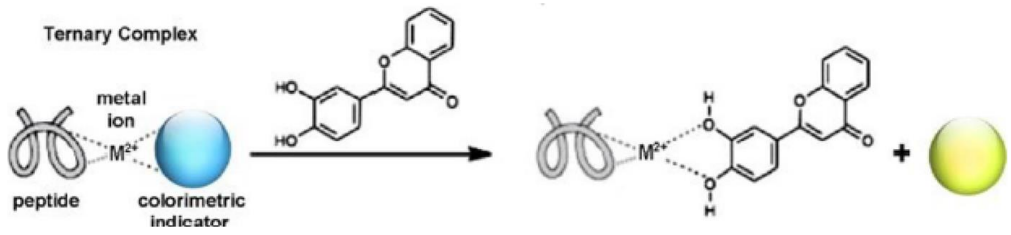


Figure 3. Mechanism of VOCs Interacting with Colorimetric Sensor Arrays (CSA) [2].

Clinically, this method allows for rapid, non-invasive diagnostics compared to traditional culture-based techniques that require longer processing times. A CSA placed near a wound can identify pathogens through the unique colorimetric pattern generated by the interaction of emitted VOCs with the sensor dyes.

3.1 Bacterium Types & Cultures

The research utilized three distinct bacterial strains: *Escherichia coli* (EC), *Pseudomonas aeruginosa* (PA), and *Staphylococcus aureus* (ATCC 6538) (SA). All protocols related to bacterial cultivation, enumeration, and experimentation followed the guidelines outlined in the ATCC Bacteriology Culture Guide [6].

Table 2. Types of Bacteria & Their Combinations.

Combination	Description
Escherichia coli (EC)	*Escherichia coli* strain with ATCC 27325 identifier
Pseudomonas aeruginosa (PA)	*Pseudomonas aeruginosa* strain with ATCC 10145 identifier
Staphylococcus aureus (SA)	*Staphylococcus aureus* strain with ATCC 6538 identifier
EC + PA	Combination of *Escherichia coli* and *Pseudomonas aeruginosa*
EC + SA	Combination of *Escherichia coli* and *Staphylococcus aureus*
PA + SA	Combination of *Pseudomonas aeruginosa* and *Staphylococcus aureus*
EC + PA + SA	Combination of all three bacterial strains

The keyword “instant” in the title comes into play when we focus our research application on Diabetic Patients because of the emergency of rapid identification in their case. The following are the relevant rationales for focusing on identifications of these strains:

Clinical Relevance: Pathogens like *Staphylococcus aureus (SA), Escherichia coli (EC), and Pseudomonas aeruginosa (PA)* are the main culprits behind wound infections, bloodstream infections, and respiratory illnesses. Due to its antibiotic resistance, *S. aureus* is a major cause of infections of the skin and soft tissues and a serious problem in hospital settings [41]. A common cause of septicemia and urinary tract infections, E. coli is a significant Gram-negative bacterium that raises morbidity and mortality [42].

The reason for including the Combinations of the Strains relates to poor immunity, and poor circulation in diabetes patients, especially in chronic lesions like diabetic foot ulcers. So, polymicrobial infections are frequent and interactions between pathogens like *Pseudomonas aeruginosa, Escherichia coli, and Staphylococcus aureus* change their metabolic outputs and increase drug resistance. While *S. aureus and E. coli* contribute by producing toxins and forming persistent biofilms, *P. aeruginosa* generates biofilms that shield co-infecting bacteria [41], [43]. By simulating the complexity of real-world infections, these bacterial combinations guarantee that detection techniques can precisely spot overlapping VOC signatures for better diagnosis.

IV. INTEGRATION OF DEEP LEARNING FOR SENSOR DATA

The use of deep learning entered the equation when rich data was being generated by each experimental trial and this information was mathematical with a lot of variances across the different species of bacterial species. Since this was an unsupervised learning problem, the logic was born when a PCA analysis was done of some data points (RGB values), and fine clusters were created (7 clusters) [1]. At this point, the process had to be automated and integrated with a learning model that has an input of an image and an output of a class of bacteria for our clinical application. A decent amount of literature was perused for learning strategies and algorithms; however, the model that we selected finally was, the ANN (Artificial Neural Network).

The RGB data produced by sensor arrays can be classified using a variety of machine-learning algorithms. Although Support Vector Machines (SVMs) have been used to classify bacteria with some success and are effective with high-dimensional data, they struggle with bigger, more complex datasets and need careful kernel selection [16]. Another viable approach, K-Nearest Neighbors (KNN), is good at handling non-linear interactions, but its scalability for real-time applications is limited by its computing cost as dataset size grows [16]. When paired with dimensionality reduction approaches, Linear Discriminant Analysis (LDA) and its hybrid versions have demonstrated good performance in bacterial species discrimination [1]. These techniques might not, however, adequately convey the complex spatial characteristics present in RGB image data.

Neural Networks were our final choice because of their prowess in processing and gleaning pertinent characteristics from visual input. Among Neural Networks, Convolutional Neural Networks (CNNs) can detect minor patterns and variations in the RGB values released by bacterial samples because they are especially well-suited for identifying spatial hierarchies within images. CNNs can achieve excellent classification accuracy, as previous research has shown;

some studies have shown up to 95% success in identifying bacterial strains from sensor data [1], [16]. In Section 6.4, the data distribution and total experimental trials amount to 235 with an inhomogeneous distribution across the selected bacterial strains and their combinations. Further in the report, CNN will be trained on this data and contrary to similar literature works, the accuracy in our case will reach 25%. The limitations, reasons, and improvements are discussed in Section 13.2 of the report.

4.1 Importance of Data Integrity & Standard Imaging Conditions

While many trials were processed and images were clicked for every time-stamped result, the VOC data captured on the Colorimetric Sensing Arrays showed various results based on the culture medium (Agar, LB-Broth, Porcine Skin), different lighting conditions, different types of bacteria species. As a result, the imaging data and the RGB data had to be organized and saved properly to prevent any discrepancy from entering the system.

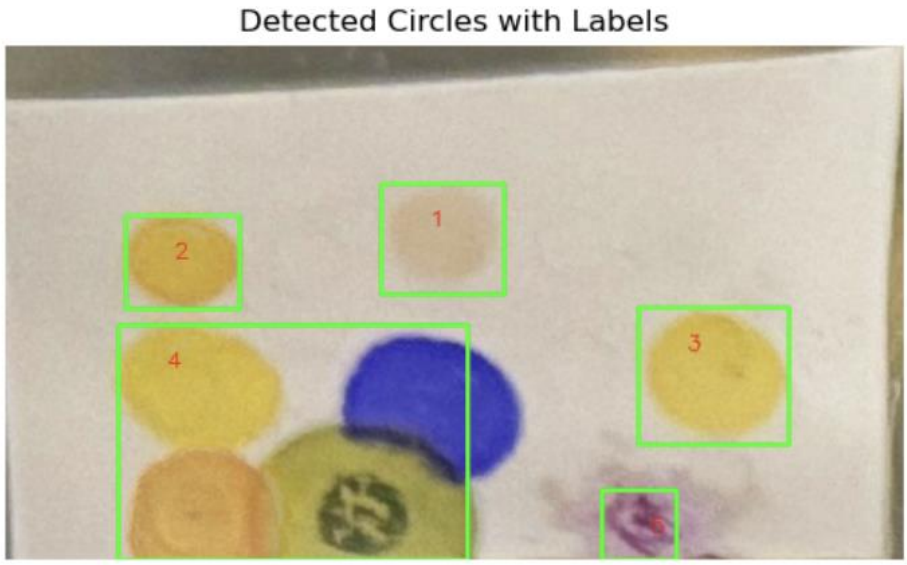


Figure 4. Incorrectly Placed Sensor Detection.

An example of incorrect imaging data in Figure 4 can include capturing images of the experimental results from inconsistent angles or the sensors placed too close to each other. With different mediums in the picture, every classification had to be organized and not inter-mixed with other results.

V. LITERATURE REVIEW: EXISTING METHODS & ANALYSIS

A variety of analytical techniques are used in the identification of bacteria utilizing volatile organic compounds (VOCs), each with its own set of benefits and drawbacks. Because of its excellent sensitivity and accuracy, gas chromatography (GC), especially when paired with mass spectrometry (GC-MS), is commonly considered the gold standard for VOC analysis. VOCs with molecular weights ranging from 2 to 1,000 Da can be analyzed by GC-MS, which has shown promise in applications like disease screening, environmental monitoring, and food inspection [17]. Its use in complicated sample analysis has increased since the advent of two-dimensional GC-MS, which includes a two-step separation (apolar and polar phases) and greatly improves the spatiotemporal resolution [18]. However, GC-MS's viability for real-time, on-site monitoring is limited by its need for a large amount of sample preparation, knowledgeable operators, and high operating expenses [17], [19].

The introduction of direct injection (DI-MS) techniques, which do not require considerable sample separation, has also advanced mass spectrometry (MS). By directly introducing gaseous analytes into the system, this technique streamlines the analytical procedure and permits quick detection [20]. Trace VOC analysis is made exceptionally possible by methods like proton transfer reaction-MS (PTR-MS) and selective ion flow tube-MS (SIFT-MS), which may attain detection sensitivities as low as parts per trillion by volume [21]. Notwithstanding these advantages, MS methods are nonetheless constrained by their reliance on tools and environments, which hinders their general applicability in standard clinical or biomanufacturing contexts [22].

With the evolution of VOC detection technology, electronic noses (E-noses) have provided an alternative method for VOC detection by using arrays of sensors sensitive to VOCs. These devices generate digital "fingerprints" that enable the identification and measurement of gas mixtures by employing clustering and classification methods such as Principal Component Analysis (PCA) and Linear Discriminant Analysis (LDA) for data interpretation [23]. However,

conventional E-noses face limitations that include sensor drift, sensitivity to environmental changes, and reduced specificity, which can compromise their reliability [24]. Additionally, some sensor types, such as metal oxide semiconductors, require high operating temperatures, leading to increased power consumption [25].

Advancements in sensor technology have led to the development of optoelectronic noses (OE-noses), which combine optical sensor arrays with chemo-responsive colorants. These sensors detect VOCs through observable colorimetric or fluorometric changes, responding to interactions such as proton transfer and hydrogen bonding [26]. Due to the high dimensionality provided by optical sensing, OE-noses can achieve excellent sensitivity (down to parts per billion) and effective discrimination among similar analytes, making them suitable for real-time monitoring in bioengineering applications [26], [27]. This method circumvents some of the drawbacks of conventional E-noses, such as non-specific interactions and basic physical adsorption [28].

In our research, we leverage Neural Networks in combination with colorimetric sensor arrays (CSAs) to enhance VOC data processing. CNNs address the limitations of conventional statistical techniques like PCA and LDA by excelling at processing non-linear and hierarchical data [29]. By incorporating CNNs, we improve the precision and flexibility of bacterial categorization, enabling automated, real-time analysis suitable for complex environments where traditional methods may fall short [17], [30]. This approach bridges the gap between practical but limited performance seen with E-noses and optical arrays and the more sophisticated yet costly technologies like GC-MS [27].

VI. METHODOLOGY

The stepwise process for making this research successful involved different steps from experimentation, hit, and trial to extensive analysis. The detailed methodology is given below:

6.1 Agar and Broth Study

First, we cultured the bacteria in LB Agar and LB Broth to produce some images and results and saw the interaction of VOC arrays. To establish bacterial populations for the study, all strains were grown in Agar Medium (agar) containing LB Agar (BD 244520) and Broth Medium (broth) containing LB Broth (BD 244620) in an incubator set at 37°C and subsequently used as needed [7], [8]. The initial inoculation population for each strain was approximately 1.5 x10^8 CFU/ml bacteria [9]. Bacterial concentration was measured in a liquid growth medium. Using a spectrophotometer plate reader (Tecan, Model: Infinite 200 PRO®, Switzerland), various dilutions of the bacterial cultures were examined for their optical density at 600 nm [10]. These readings were then compared to a standard MacFarland sample to determine the bacterial concentration [11].

Additionally, this research examined the impact of combining different bacteria to replicate a polymicrobial wound infection environment. The study assessed the sensor's effectiveness in situations where two or three bacterial strains coexisted, enabling observations of their interactions, potential rivalry, and diverse growth patterns [12], [13]. The research contrasted Agar and Broth media to determine how the growth substrate affects the bacterial VOC profile [14]. To test the sensor, researchers placed a small Petri dish containing either agar or broth inside a larger Petri dish, with the sensor attached to the inside of the larger lid [15].

6.2 Porcine Skin Study

To evaluate the sensor's ability to detect bacteria through their VOC signatures, we performed additional tests on ex vivo porcine skin grafts from ears. We acquired fresh pig ears from a local butcher in Victoria, BC, Canada, and transported them on ice immediately. The skin was cleaned

with warm water in the lab, treated with hydrogen peroxide (10% for 5 min, followed by 3% for 10 min), and rinsed with warm Dulbecco's phosphate-buffered saline (DPBS; Gibco, Thermo Fisher Scientific Inc.) [27], [6]. Uniform skin sample disks were created using a 6 mm diameter surgical punch. These samples were immersed in warm Dulbecco's Modified Eagle Medium (DMEM; Gibco, Thermo Fisher Scientific Inc.) containing 3% gentamycin and 1% amphotericin at 37°C for 18 h [6], [11]. After washing twice with DPBS, they were placed in warm DMEM (antibiotic-free) at 37°C for 4 h to allow any remaining antibiotics and antifungal agents to dissipate.

The prepared skin samples were then positioned in custom-designed trans wells containing 800 µl of plain DMEM at the bottom, with the top exposed to air, creating an air-liquid interface [15]. [22]. These trans wells were fabricated using a standard fused deposition modeling (FDM) 3D printer with certified food-grade PLA+ (polylactic acid polymer) filaments. Post-processing, they were autoclaved and kept in sealed bottles to maintain sterility for testing [15], [19]. On the test day, they were attached to each Petri dish using double-sided tape in a biosafety cabinet.

To replicate an infected wound on a live porcine skin sample, we introduced bacteria in similar populations and media onto the skin as a droplet [7], [9]. A wound of comparable depth and size was created using a surgical knife, allowing the bacteria to enter and initiate infection. The prepared wound models in Petri dishes were then transferred to an incubator and kept at 37°C .

6.3 Imaging Process

Images were captured using a conventional digital camera under consistent lighting to ensure uniform environmental conditions. The captured images were then analyzed using Fiji software (ImageJ, Version 1.54k) to extract RGB values from 9 designated spots for further examination. As described in Section 4.1 about the importance of data integrity for our study, the sensor arrangement and the lighting conditions had to be standardized to maintain consistent imaging protocols across the entire process. Every sensor dye has a diameter of 2mm, and they are separated by 4 mm to make the entire array. All the images are captured in three different lighting

conditions – low, ambient, and bright light. Based on different lighting conditions, the images corresponding to a class of bacteria have to be evaluated to maintain consistency in results to mimic the situations for capturing the wound by the app (MAK-1) we developed. Future testing will entail the three lighting conditions and produce the results for the same class in every lighting condition to check the accuracy of predictions for the same class of bacteria.

6.4 Data Analysis and Data Cleaning

The dataset had a total of 235 experimental trials done throughout the course of the project which can be distributed into three bacteria classes – EC, SA, PA and their combinations in three mediums – Agar, Liquid Broth, and Pig Skin. Their distribution can be seen below in the form of a pivot chart. An emphasis should be placed on the data distribution, as we didn't want a learning algorithm to be overtrained for one set of bacteria. We had limited trials for pig skin, but further in the analysis section, we will prove the accuracy of clustering results in this medium.

Table 3. Data Distribution (Up-till today).

	Number of Trials
EC	**29**
Agar	12
Liquid	12
Pig Skin	5
EC+PA	**38**
Agar	23
Liquid	10
Pig Skin	5
EC+PA+SA	**38**
Agar	18
Liquid	15
Pig Skin	5
EC+SA	**49**
Agar	23
Liquid	21
Pig Skin	5
PA	**28**
Agar	13
Liquid	10
Pig Skin	5
SA	**29**
Agar	13
Liquid	11
Pig Skin	5
SA + PA	**19**
Agar	19
SA+PA	**5**
Pig Skin	5
Grand Total	**235**

As for the Definition of Data, for each row, we have 27 columns composing the RGB values of the 9 Selected Dyes based on the evaluation of results (Section 9.3). Each Column corresponds

to the RGB values of each dye. For Example, for the first dye, the columns are Dye1_R, Dye2_G, and Dye3_B.

6.5 Algorithm Development & Selection

The above dataset was utilized and pre-processed to train some classification machine learning algorithms to prove the viability of our concept and how this process can be automated and implemented with learning algorithms. In the given study the following algorithms were tested in the given order to find connections and evolutions as we jump from one study to another.

1. Clustering Techniques with PCA analysis: For the clustering, 5 samples in each medium are plotted to show valid clusters in space after doing some Principal Component Analysis (PCA) on the features of the dataset. Based on the valid clustering, the use of mathematical models became valid for this study. This shows that data holds information about the bacteria species.

2. KNN & RF, and their combination: The labelled dataset was used for this study. For classification tasks, the K-Nearest Neighbors (KNN) and Random Forest (RF) models are flexible, non-parametric machine learning methods. RF employs an ensemble of decision trees to improve prediction stability and accuracy, whereas KNN classifies based on the majority class of the closest data points. By combining the advantages of both models, they can produce an ensemble classifier that is more resilient.

3. Support Vector Machine (SVM): Finding the best hyperplane to divide data points of several classes is the goal of the SVM, a potent supervised learning technique. It is efficient when there is a distinct margin of separation between classes and performs well in high-dimensional spaces.

4. Artificial Neural Network (Deep Learning Technique): Inspired by the human brain, an Artificial Neural Network (ANN) is a deep learning model that can identify intricate patterns in data. It can handle complex categorization issues and is made up of interconnected layers of nodes (neurons) that can learn non-linear correlations.

VII. DESIGN FOR THE PIPELINE (DFD) for MAK – 1

The entire research was developed into a Web-Based Application as well as called MAK – 1. Link for the Application - https://mostafasapp01.wl.r.appspot.com.

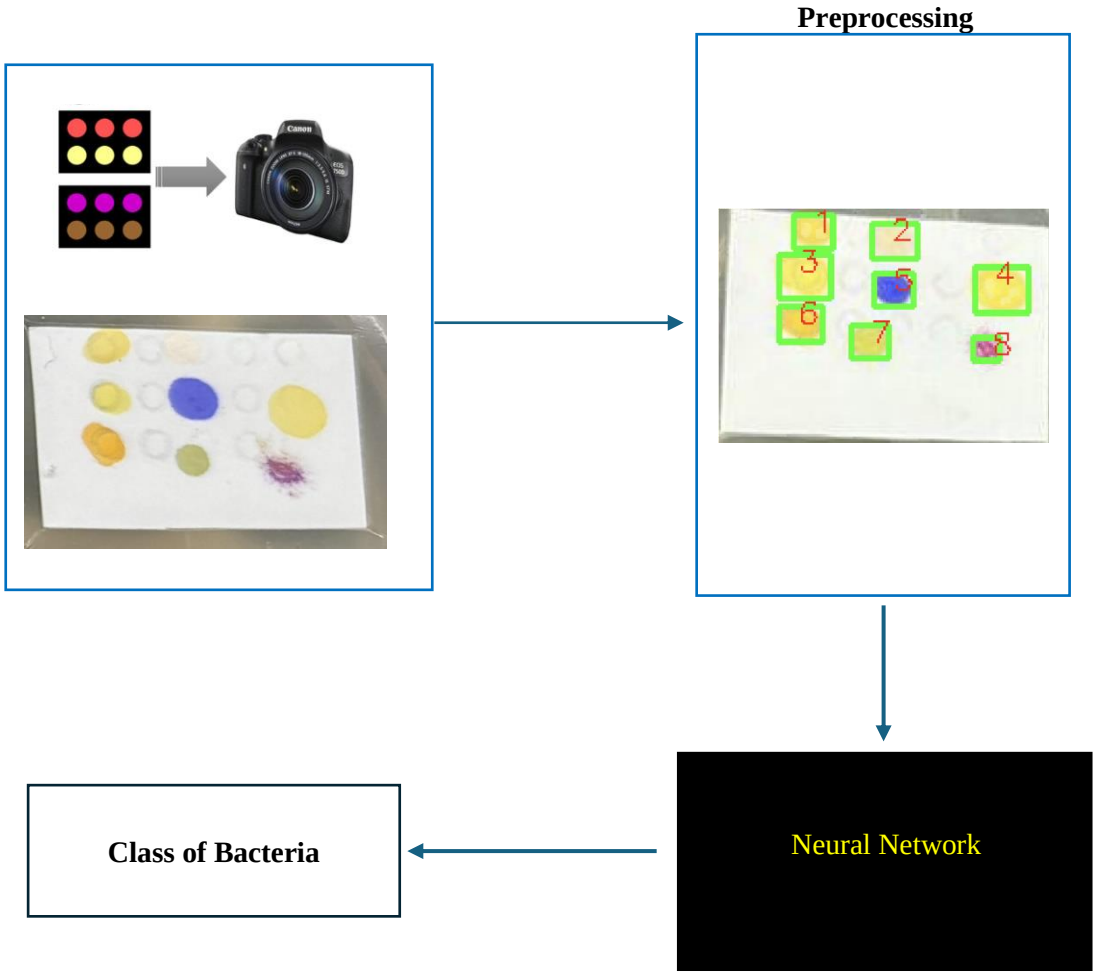


Figure 5. Data Flow Diagram for the Pipeline.

1. In Figure 5, the first step is to capture the Colorimetric Sensing Array after a period of 1-2 hours to allow the VOCs to produce chemical changes causing the shift in the RGB values.
2. Secondly, the captured image will go to the pre-processor to extract the RGB values.
3. The ANN or CNN (the one with higher accuracy) trained on our data will make the decision to give the results.

VIII. DEEP LEARNING MODELS ARCHITECTURE

8.1 Artificial Neural Network (ANN)

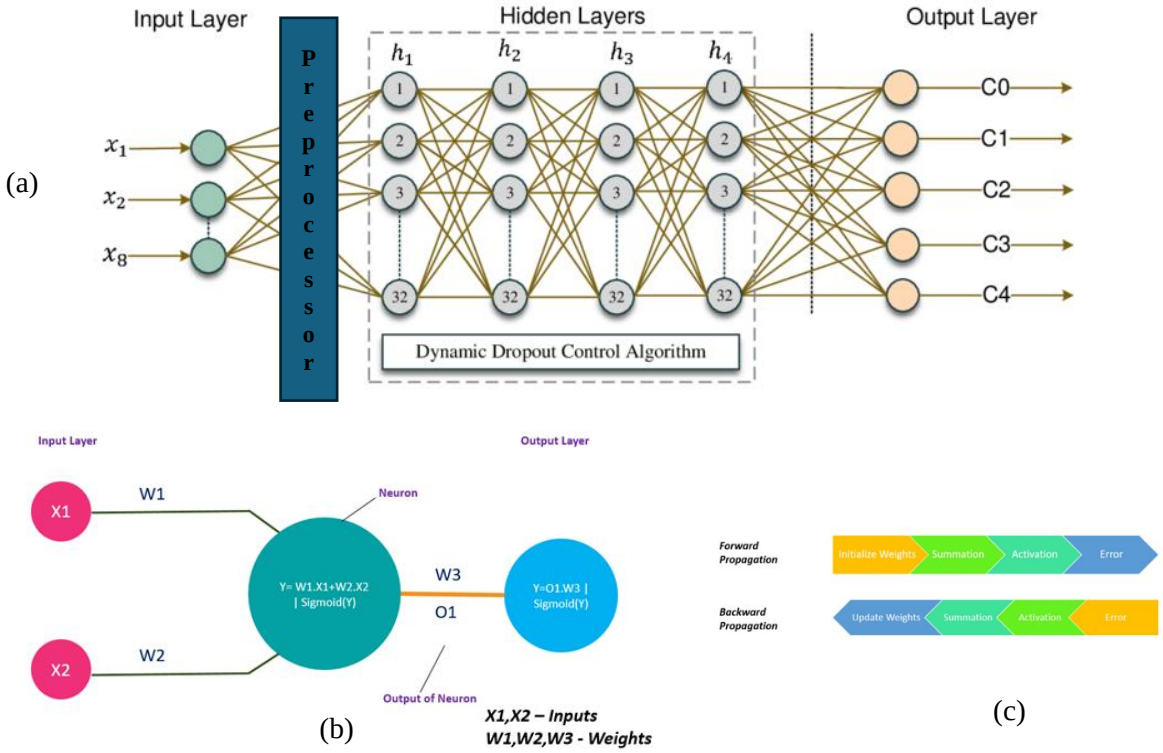


Figure 6. (a) Six Layer ANN Model Basic Architecture [40]; (b) Example for a Single Neuron; (c) Working Principle of ANN.

Input Layer – The Input Layer in our case receives the RGB values of the Dye spots from the Pre-Processor.

Hidden Layers – Our Model comprises 4 Hidden Layers, each layer consists of 512, 256, 128, and 64 neurons respectively.

Output Layer – Consists of only one neuron, making the final classification based on the input received by the 4th hidden layer.

Activation Function – Leaky ReLU = f(x) = max(0.01*x, x)

Working Principle – The ANN adjusts the weights in forward and backward propagation (Epoch) based on the loss function minimization results.

8.2 Convolutional Neural Network (CNN)

Given below is the architecture that will be integrated into the Front-End Application after doing trials and testing predictions for the CNN model performance.

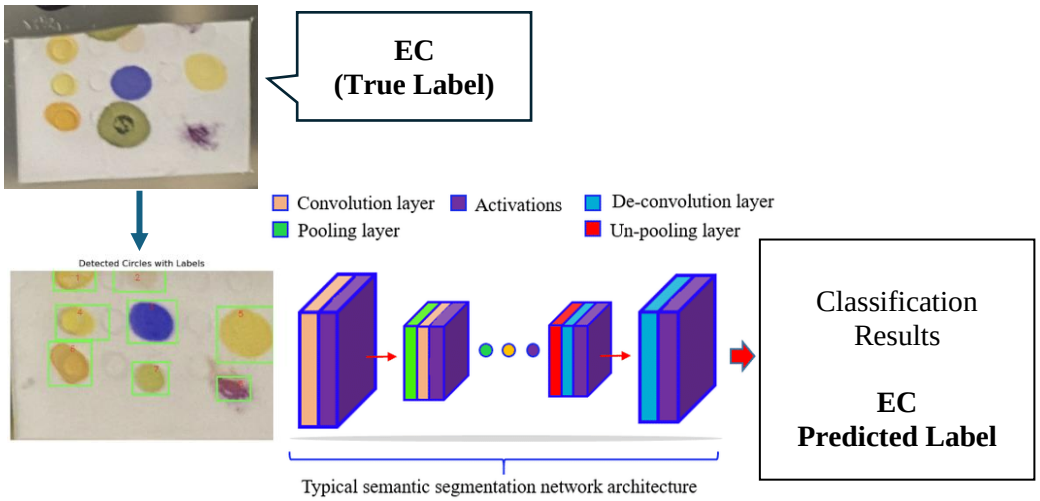
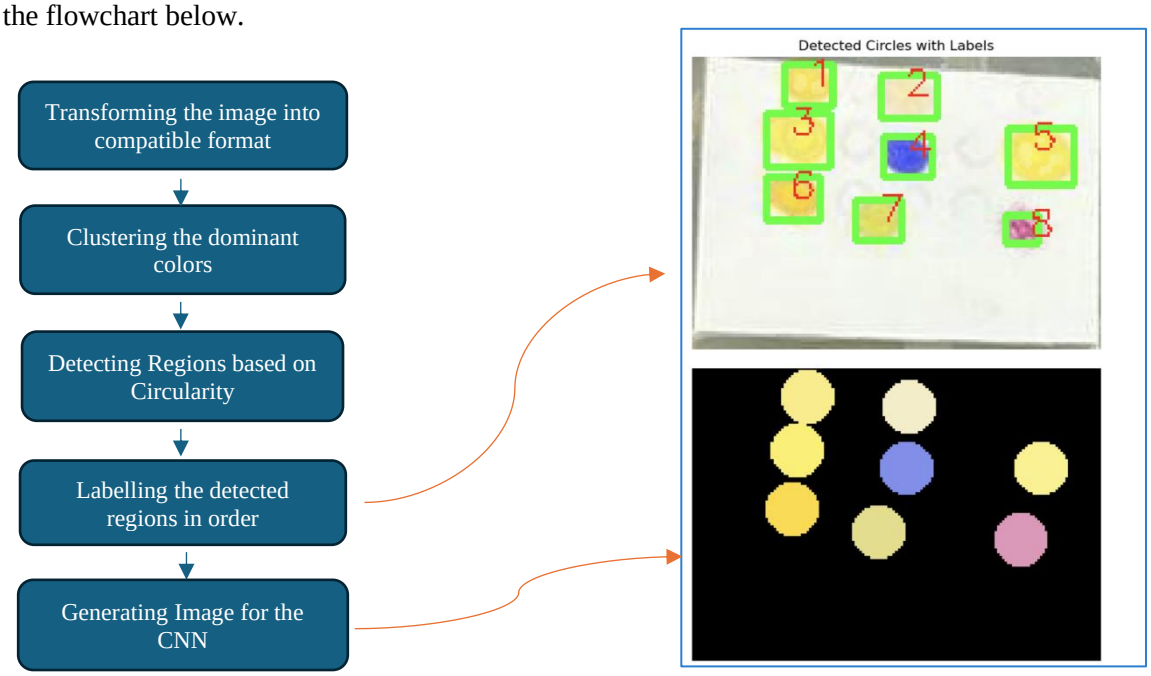


Figure 7. CNN Model Architecture

The concept is to capture the image and introduce absolute automation from end to end. This level of abstraction required an accurate algorithm to preprocess the image to extract the RGB values and highlight the regions of interest as highlighted above. These Computer-Generated Images are the training images for the CNN model. The algorithm for the preprocessor is given in the flowchart below.



IX. ANALYSIS OF RESULTS

This section shows how we set our pipeline to evolve from the data and use it to increase the accuracy of predicting the class of bacteria species from imaging data. First, the clustering of the data points was explored to prove that we can use mathematics for processing the CSA data. As the data got rich in information with time, we started using supervised learning strategies to test the accuracy of the models. Given below are the results for each strategy employed and the evolution of accuracy as we go ahead.

9.1 HCA, PCA, and Clustering Results in Agar and Broth Medium

For conducting the HCA, and PCA, the delta RGB values (before and after exposure) were calculated from the dye spots using the Pre-Processor of the MAK-1. The Euclidean distance for the dye RGBs was evaluated in Microsoft Excel as mentioned in Section 10.3. For statistical analysis, the data was imported into GraphPad Prism (version 10). Multivariate analyses, including hierarchical cluster analysis (HCA) and principal component analyses (PCA), were performed. The HCA was conducted using Ward's method with Euclidean distance as the measure.

Bacterial growth, reproduction, and metabolic processes are largely influenced by the culture medium. Research has shown that the composition of volatile organic compounds (VOCs) produced during bacterial metabolism varies depending on the growth medium. To assess whether different culture media affect the sensor's effectiveness, we tested its capacity to correctly identify bacteria cultivated in two widely used in vitro media: liquid LB broth and solid agar. Additionally, we explored various combinations of the three bacterial species to determine if the sensor could detect distinct patterns among individual species. This study is particularly relevant because wound infections typically involve multiple bacterial species rather than a single type. As a result, we not only analyzed EC, SA, and PA separately but also prepared and evaluated mixtures of EC+PA, EC+SA, SA+PA, and EC+PA+SA.

LB agar culture medium has been widely utilized as a substrate for bacterial growth in research examining bacterial VOC profiles. As depicted in Figures 9a and 9b, an agar medium was utilized to fine-tune sensor parameters, and subsequently employed to assess the sensor's ability to categorize bacteria based on VOC emission analysis, as shown in Figure 9a. The figure presents clustering analysis outcomes for five replicates of various bacteria grown on the agar medium, along with uninoculated agar as a control. These were effectively sorted into seven distinct clusters, separated by a considerable Euclidean distance from an additional cluster comprising five replicates of the control group (an uninoculated agar plate). The figure also includes color difference maps for each cluster to aid visual interpretation. The sensor demonstrated 100 percent accuracy in classifying the bacteria into separate clusters based on their type. To corroborate the HCA findings, Figure 9b displays the PCA results for the identical dataset, which achieved 100 percent accuracy in grouping all eight clusters, thus confirming the HCA results. Figure 9b showcases the PCA biplot, elucidating the magnitude and nature of each dye spot's impact on bacterial classification in the PCA analysis.

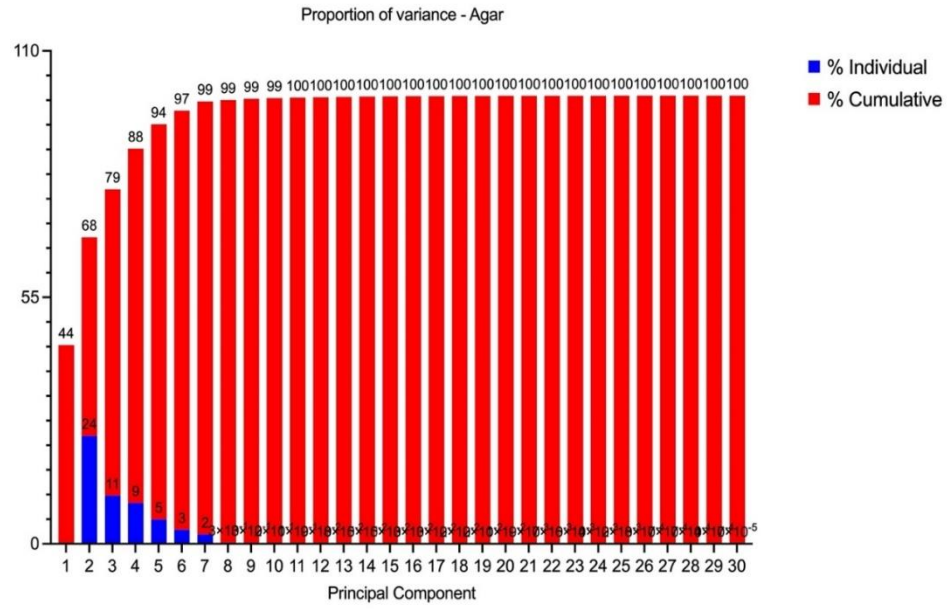


Figure 8. Principal Component Analysis of Dye Sensors Values in Agar.

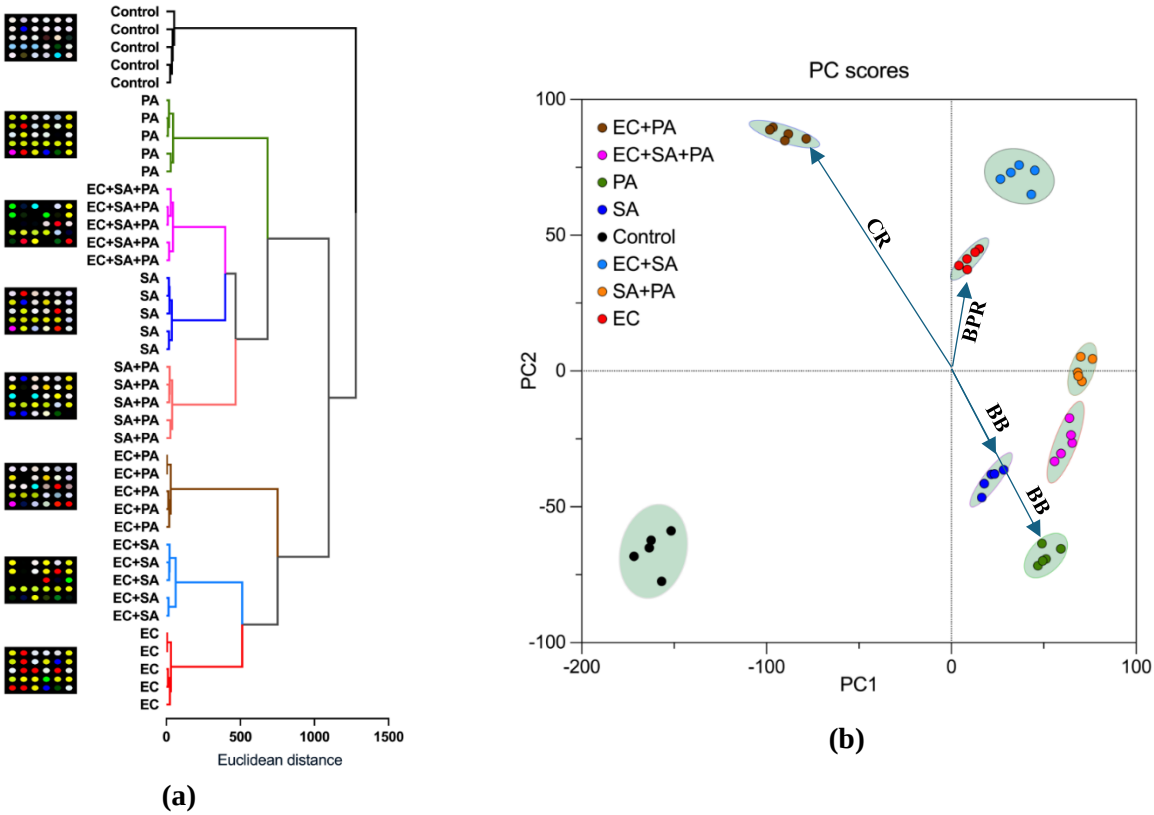


Figure 9 (a) Hierarchical Cluster Analysis in Agar [45].

(b) Principal Component Analysis Clusters in Agar (Bipolar Plot), the arrows indicate dye names [45].

A comparable investigation examined the color difference maps for bacteria cultivated in liquid broth media Figure 11a, offering a thorough visual depiction that forms eight distinct clusters with flawless grouping accuracy, aligning perfectly with the outcomes from agar cultivation. This uniformity across various growth media showcases the classification method's reliability. The substantial separation between each category is also visually evident in the color difference maps. As consistent with the agar results, the PCA findings for Liquid Broth are in Figure 11b. Additionally, the PCA biplot analysis illustrated in Figure 11b reveals that most dyes exhibit a negative correlation with PC1, while some show a positive association with PC2. The considerable vector magnitudes (blue arrows) in Figures 9b and 11b indicate that principal components are greatly influenced by the dyes used in this context, highlighting the appropriateness of the chosen dyes for the CSA.

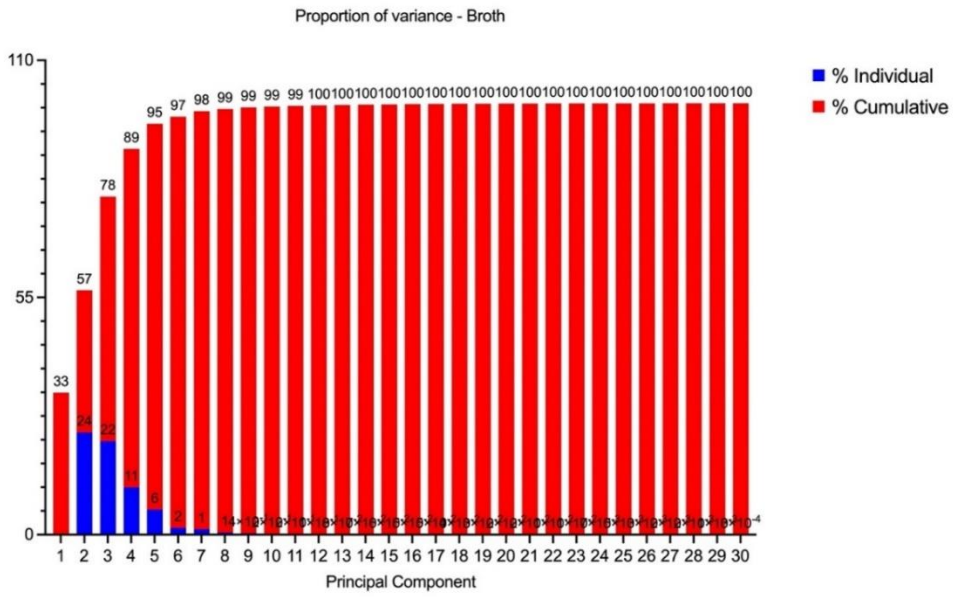


Figure 10. Principal Component Analysis of Dye Sensors Values in Liquid Broth.

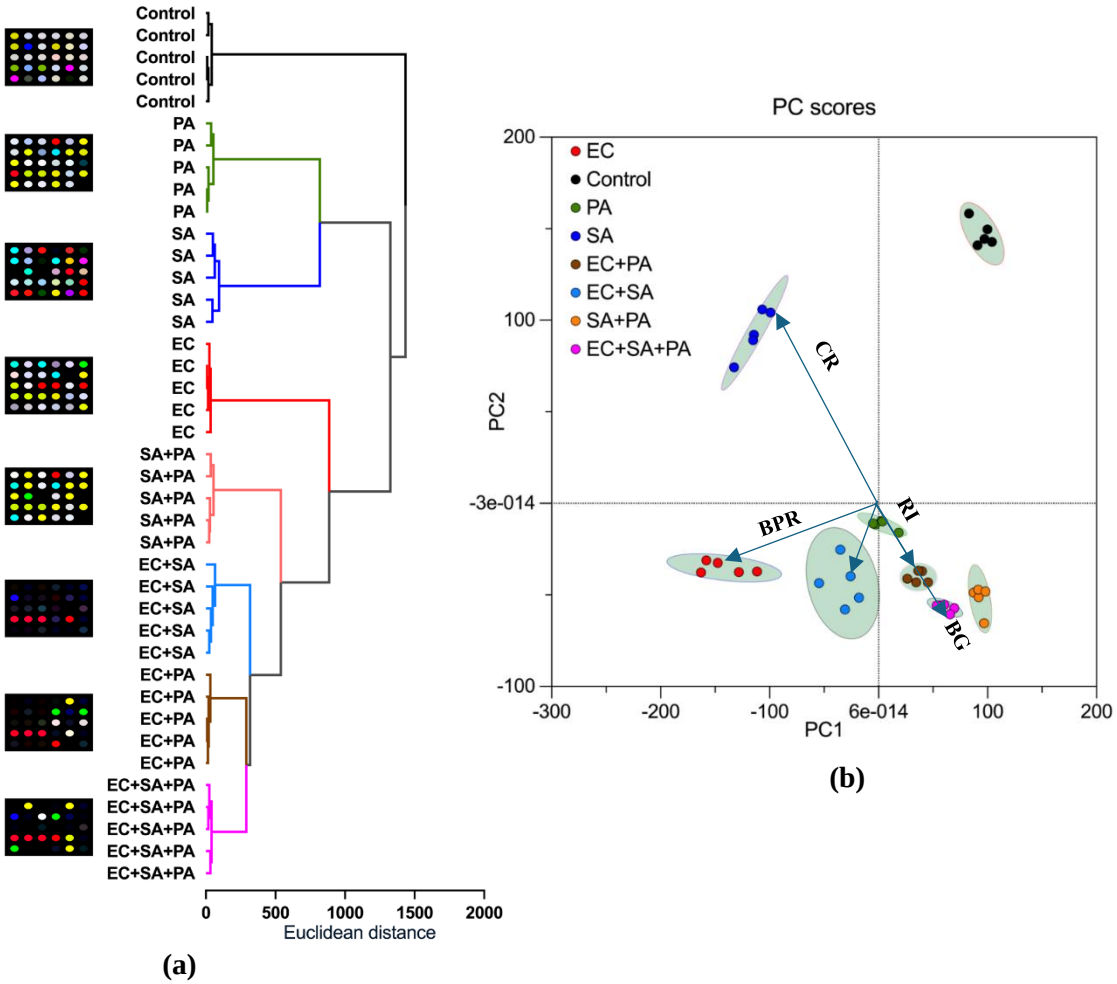


Figure 11. (a) Hierarchical Cluster Analysis (HCA) in Liquid Broth [45]; (b) Principal Component Analysis Clusters in Liquid Broth (Bipolar Plot), the arrows indicate dye names [45].

Furthermore, the PCA cumulative variance percentage, shown in Figures 8 and 10, provides crucial insights into the data variability captured by each principal component for both agar and broth cultures. This information aids in evaluating the effectiveness of dimensionality reduction and the importance of the observed groupings. The findings from both agar and broth experiments demonstrate that the sensor can identify bacteria based on their VOC profile, regardless of the culture medium. Notably, this capability extends to different materials and various states, including liquids and solids, a feature previously unreported for any other VOC sensors.

9.2 PCA & Clustering Results in Porcine Skin

Expanding upon the encouraging outcomes of the sensor in conventional in vitro culture media, we employed porcine ex vivo specimens as a growth substrate to simulate human skin. We created a simulated wound on these samples using a scalpel and introduced bacteria to mimic an infection. Consistent with the findings from agar and LB broth cultures, VOC-CSA accurately distinguished between bacterial types with 100% precision as shown in Figure 13b.

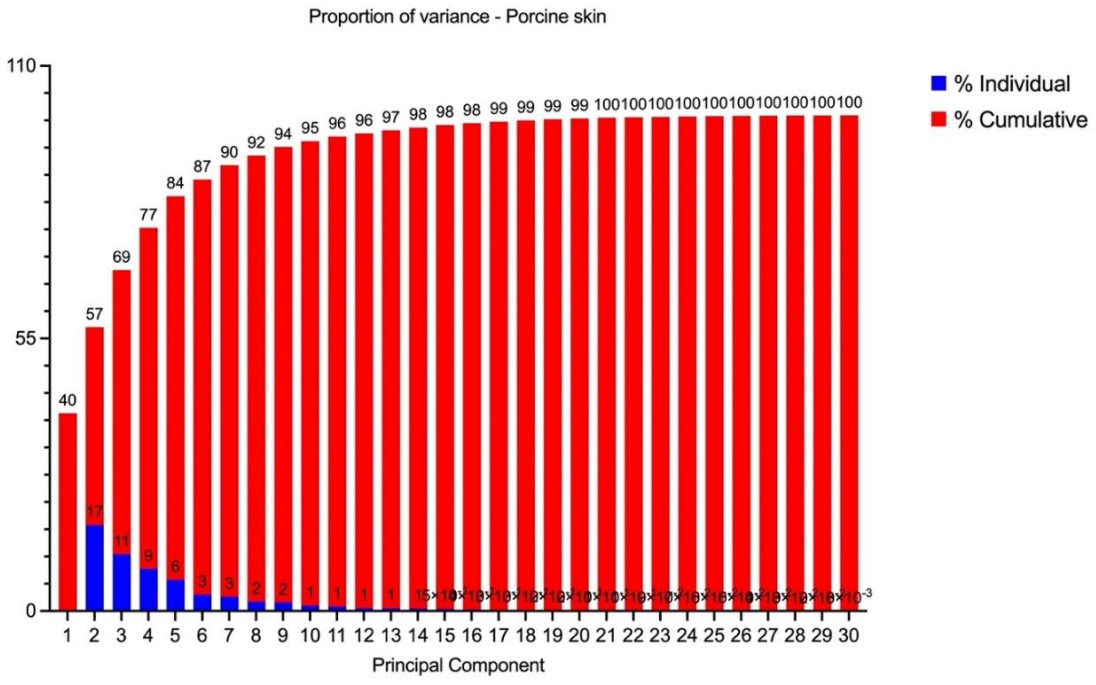


Figure 12. Principal Component Analysis of Dye Sensors Values in Porcine Skin.

Because of the striking similarities between pig and human skin, research on pig skin has opened the door to testing on human bio-printed models. The structural and functional properties of porcine skin, which is frequently employed as a surrogate in biomedical research, are similar to those of human skin. These features include similar epidermal thickness, collagen distribution, and barrier function. Because of this similarity, it is a perfect model for researching how people react to outside stimuli, particularly those measured by optoelectronic sensors.

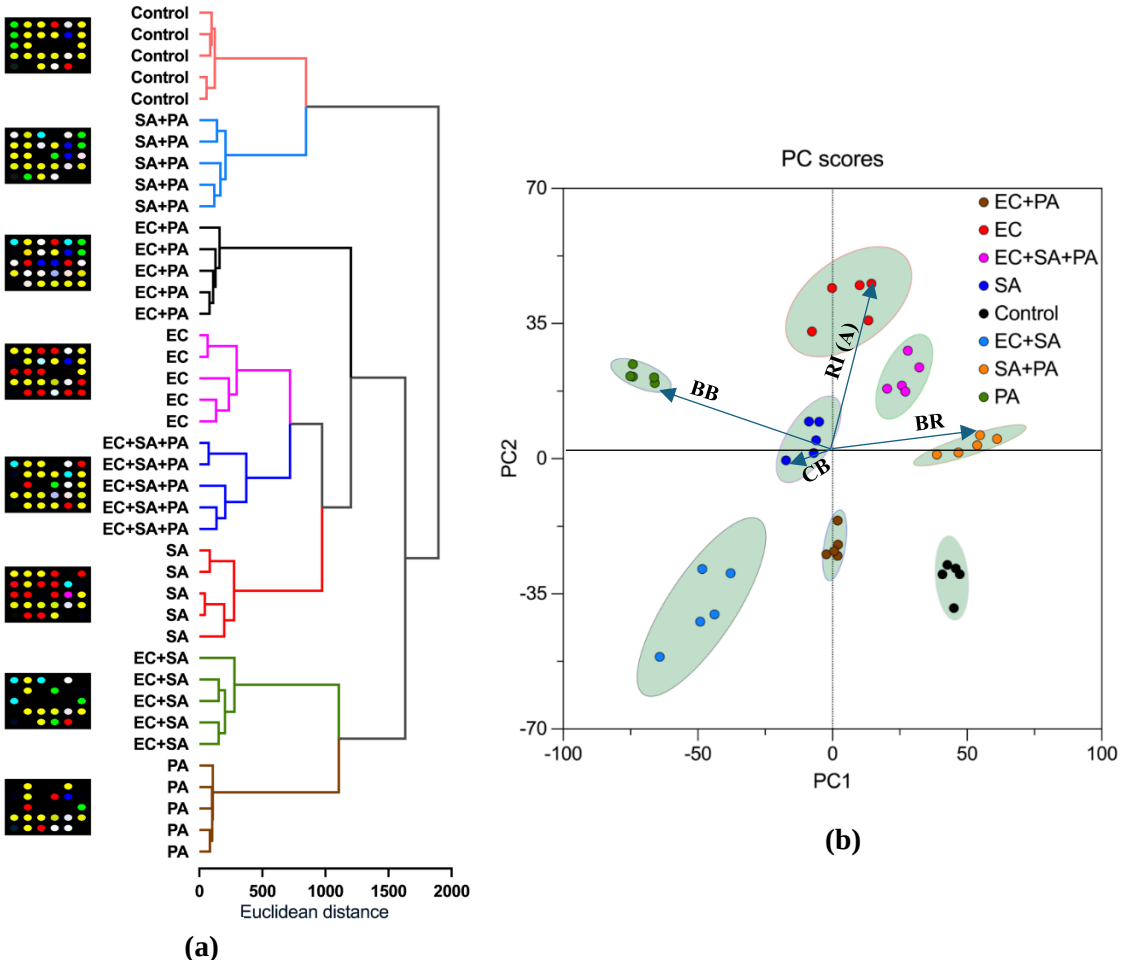


Figure 13. (a) Hierarchical Cluster Analysis (HCA) in Porcine Skin [45]; (b) Principal Component Analysis Clusters in Porcine Skin (Bipolar Plot), the arrows indicate dye names [45].

9.3 Dye Selection (Based on HCA & PCA Results)

In Figures 9b, 11b, and 13b, the arrows (blue) indicate the impact of dyes on the separated clusters in the reduced dimensional space (PC1 and PC2). Each arrow points in the direction of a cluster, demonstrating how a particular dye aids in the separation or identification of various bacterial groups according to the chemicals (metabolites) they emit. Which species or group of bacteria

the dye is most sensitive to is indicated by the arrow's length and direction. Based on the PC plots, we filtered the 30 dyes based on their contribution to the principal components (PCs) which produce at least 99% variance (figure 8, 10, 12). We get a list of a few dyes explaining the maximum contribution to explain the variance in separating the samples in PC biplots. Each dot is a sample, and every arrow represents which dye made it to differentiate from the rest based on the delta RGB shifts.

Table 4. Final Selected Dyes for the CSA.

New Spot	Old Spot	Name of Dye
1	2	Bromothymol Blue
2	3	α- Naphtholphthalein Acid
3	5	Richards Dye Acid
4	7	Brilliant Yellow Basic
5	11	Comassie Blue
6	12	Bromophenol Red Basic
7	19	Bromocresol Green Acid
8	22	Nitrazine Yellow
9	29	Cresol Red

In Table 4, we have the final selected dyes based on some PC analysis as well as discarding any inactive dyes from the 30 we had selected (the ones that show no change in RGB values). Based on the above dyes we created our sensors and developed all the further machine learning and deep learning algorithms. Explaining more about the PC analysis, when we had the delta RGB values of the 30 dyes, we ran a PC analysis (PC - Finding the recipe that contains the 30 dyes but produces maximum variance in the data). Every PC has its recipe for the 30 dyes. Theoretically, 30 PCs were assigned initially, and we saw the cumulative variance caused by all the PCs. Then when we considered that PC value where the cumulative variance became 99%. Then we saw the

contribution of the dyes in those PCs and ranked them as per their cumulative fractional contribution in every one of those PCs.

9.4 Clustering Analysis of the Selected 9 Dyes in Porcine Skin for Dimensionality Reduction

Even after doing clustering analysis on the nature of the sensor values based on the 9 selected dyes in Table 4, we could see the results to be placed in the space in the form of distinct clusters in Figure 14 suggesting the selection to produce valid results and reduce dimensionality in the data for our convenience. Going forward we developed all the analysis and results of the various AI models based on these selected dyes' RGB values.

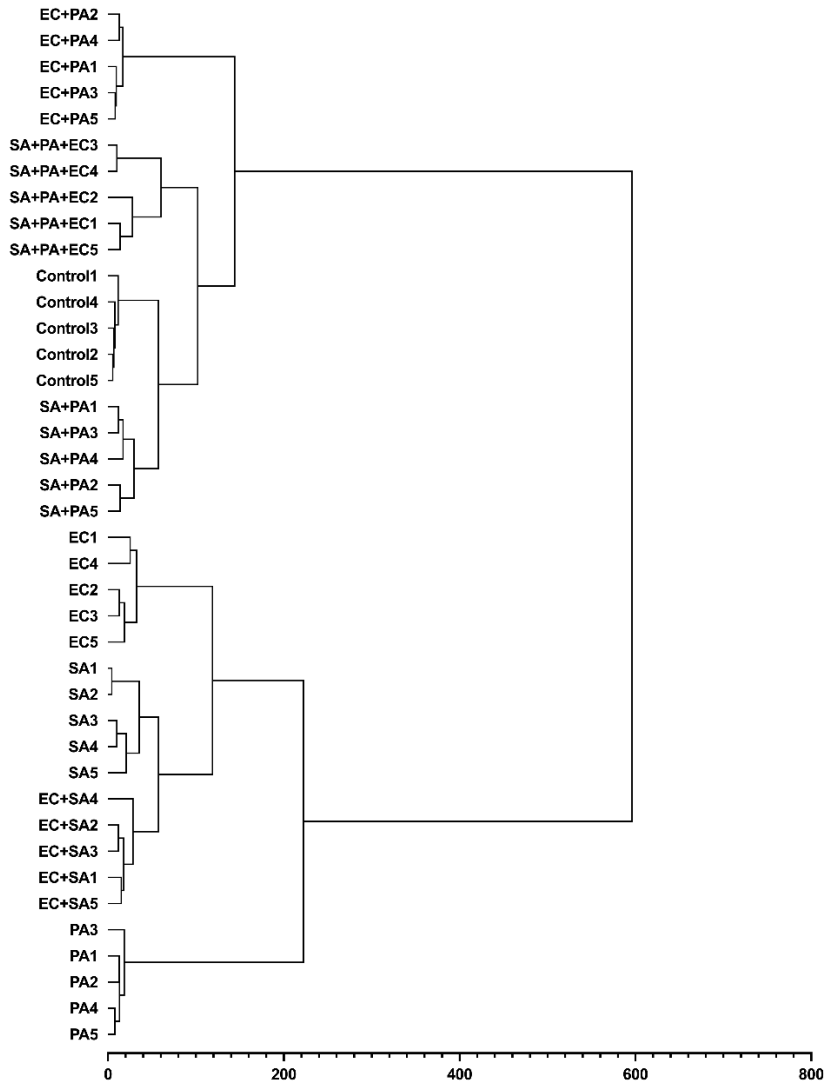


Figure 14. Hierarchical Clustering Analysis of 9 Selected Dye Data in Pig Skin.

9.5 K- Nearest Neighbor (KNN), Random Forest (RF) & Ensemble Model

Background – As KNN, Random Forest, and an ensemble model each have special advantages for our research, we applied them. Reliable predictions are produced by Random Forest, which handles complex and unbalanced data by building many decision trees and combining their output. We leverage the characteristics of both approaches by integrating them into an ensemble model.

Dataset - The KNN and RF models were trained using two datasets (one by one) to check the variance in the accuracies for each dataset. The first dataset is the normal file which has 27 columns, but for the second replicate of the models, we reduced the dimensions of the dataset to only 9 columns. Refer to Section 10.3 for more details.

KNN, RF, and Ensemble Accuracies – 38%, 45%, 44%

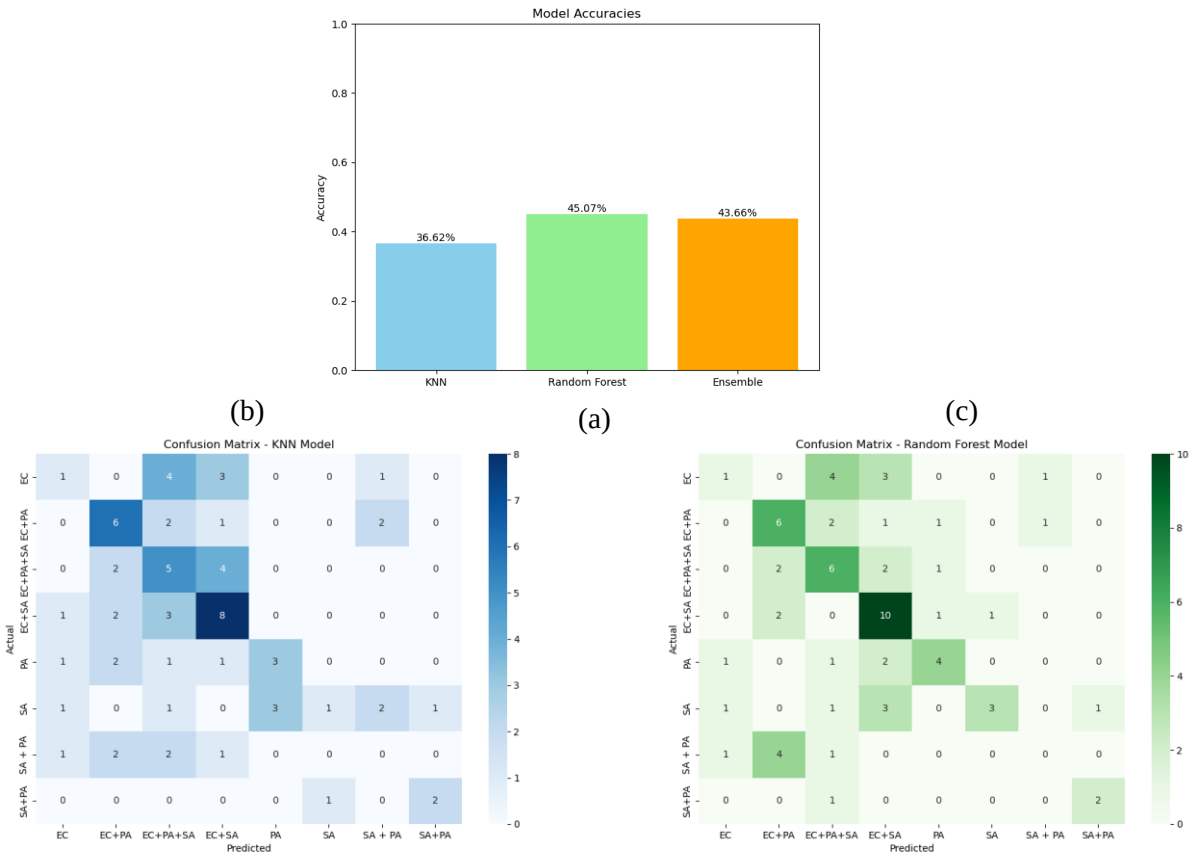


Figure 15. (a) Comparison of Accuracies for KNN, RF, and Ensemble Model on the Reduced Dimension Dataset; (b) Confusion Matrix for KNN on Reduced Dimension Dataset; (c) Confusion Matrix for RF on Reduced Dimension Dataset.

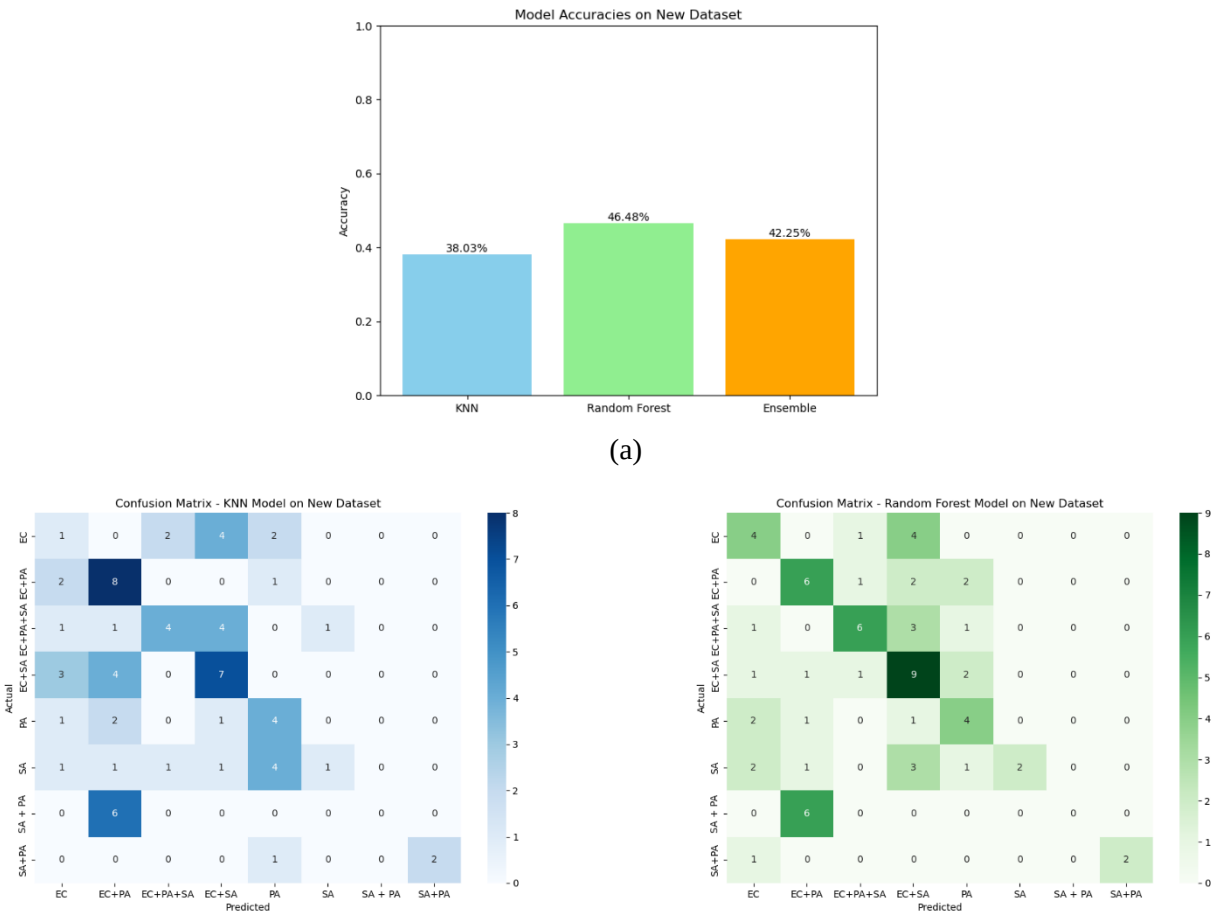


Figure 16. (a) Comparison of Accuracies for KNN, RF, and Ensemble Model on the Normal Dataset; (b) Confusion Matrix for KNN on Normal Dataset; (c) Confusion Matrix for RF on Normal Dataset.

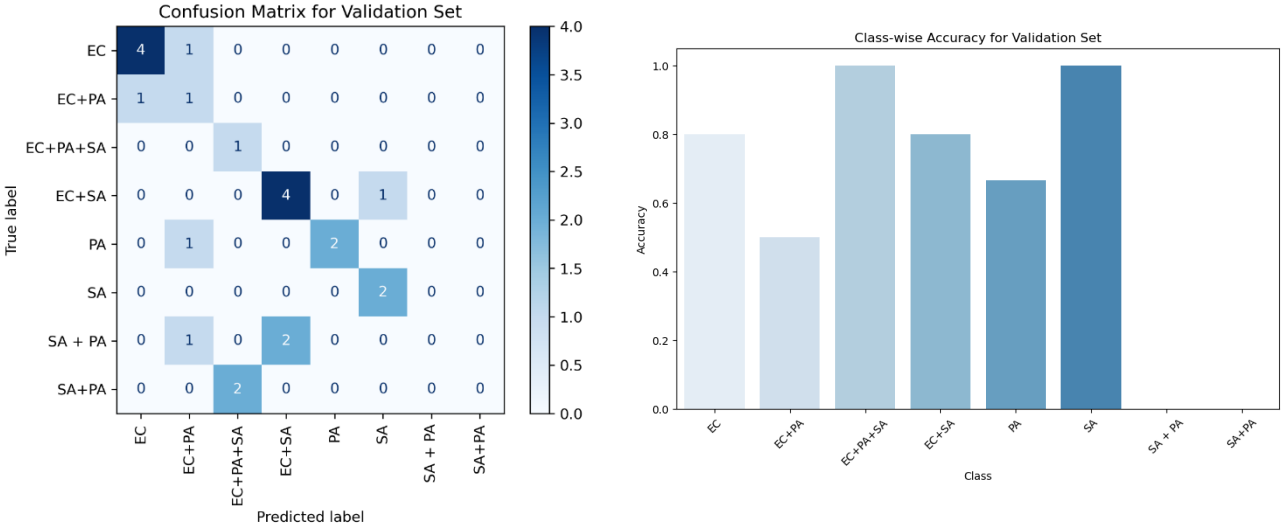
9.6 Support Vector Machine (SVM) Model

Background – Finding the best hyperplane to divide classes in a dataset is how the potent classification algorithm Support Vector Machine (SVM) operates. SVM is extremely good at handling high-dimensional data and can withstand overfitting, especially when there are more features than samples. By converting the input data into a higher-dimensional space where a linear separation is feasible, its use of kernels enables it to describe nonlinear interactions. Because of this, SVM is very adaptable to datasets with intricate patterns.

The training parameters for your SVM model are kernel='linear', random_state=42, with default C=1.0, tolerance for stopping =1e-3, and unlimited iterations.

To guarantee the reproducibility of findings, machine learning models use the random state parameter, such as random_state = 42 in your SVM model. It controls the randomness of some processes, like data splitting or shuffling, by serving as a seed for the random number generator. dependability and comparability.

SVM Model Accuracy: 61%



(a) (b)

Figure 17. (a) Confusion Matrix for the SVM model; (b) Class-Wise Accuracy Results for the SVM Model (Y-axis represents fractional accuracy).

9.7 Artificial Neural Network (ANN) Model

- Layers:
    - 512 units, LeakyReLU, Dropout(0.5), L2 regularization
    - 256 units, LeakyReLU, Dropout(0.4), L2 regularization
    - 128 units, LeakyReLU, Dropout(0.3), L2 regularization
    - 64 units, LeakyReLU, Dropout(0.2), L2 regularization
    - Output layer with softmax activation for multi-class classification.
    - BatchNormalization is applied after each dense layer.
- Optimizer: Adam optimizer (Gradient Based) with a learning rate of 0.0003.
- Loss Function: sparse_categorical_crossentropy for multi-class classification with integer-encoded targets.
- Metrics: Accuracy for evaluation during training and validation.
- Callbacks: Learning Rate Scheduler: Reduces the learning rate by a factor of 0.5 after 50 epochs without improvement in validation loss, with a minimum learning rate of 1e-6.

- Early Stopping: Stops training if validation loss does not improve for 100 consecutive epochs, restoring the best weights.
- Hyperparameters:
    - Epochs: 581, Up to 1000 (stops earlier if early stopping is triggered).
    - Batch Size for Training: 16.
    - Input Shape: The input shape for the model is based on the preprocessed training data (235).
    - Regularization: L2 regularization with a coefficient of 0.001 in all dense layers to prevent overfitting.
- ANN Model Accuracy – 91.67%

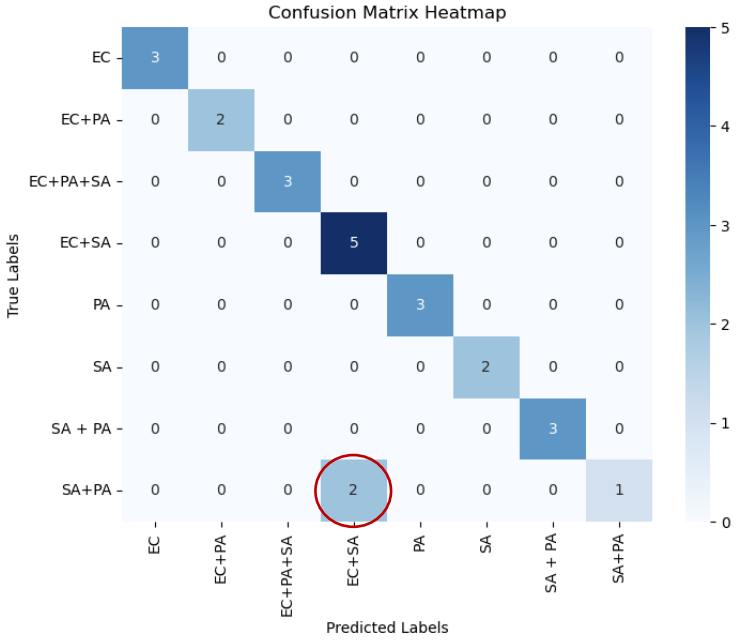


Figure 18. Confusion Matrix for the ANN Model.

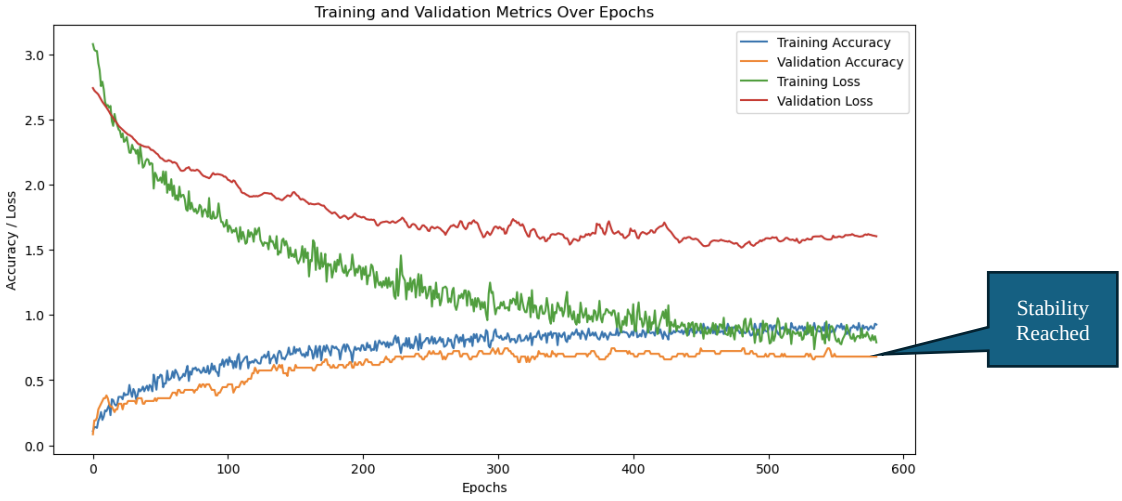


Figure 19. Training History for the ANN Model.

9.8 Convolutional Neural Network (CNN) Model

Accuracy – 25%

Reason for Less Accuracy– Size of the image's dataset (235)

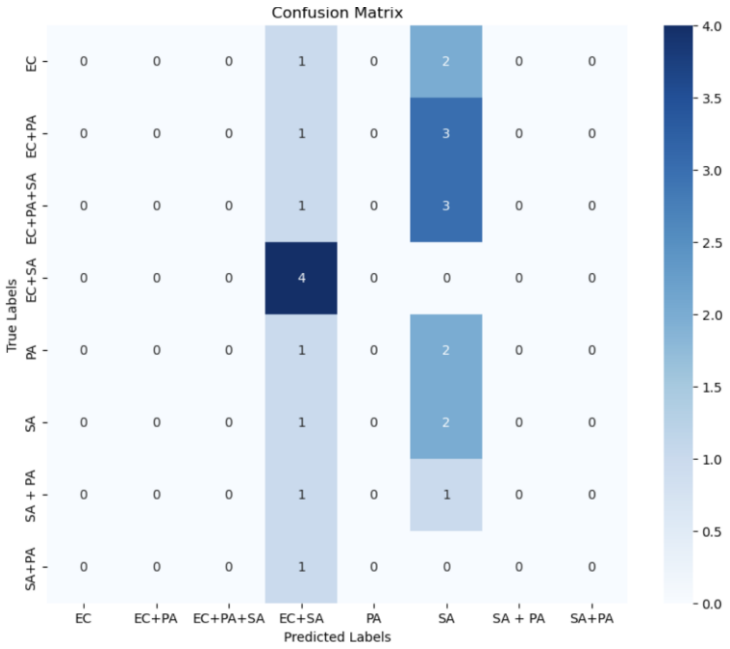


Figure 20. Confusion Matrix for the CNN Model.

CNN model has potential and hasn't ever been explored in research. Since the starting data point of our application is an image, this model can be fed that input, and any middle pre-processor can be removed from the equation to prevent any unnecessary errors in the pipeline. Going further, the goal is to expand the dataset to at least 1000 images and then try the training again. In the app, we have a section reserved for just training on new images to improve the mode accuracy. Refer to section 13.2 for more details.

X. EVALUATION OF RESULTS

10.1 Accuracy Comparison for the Models

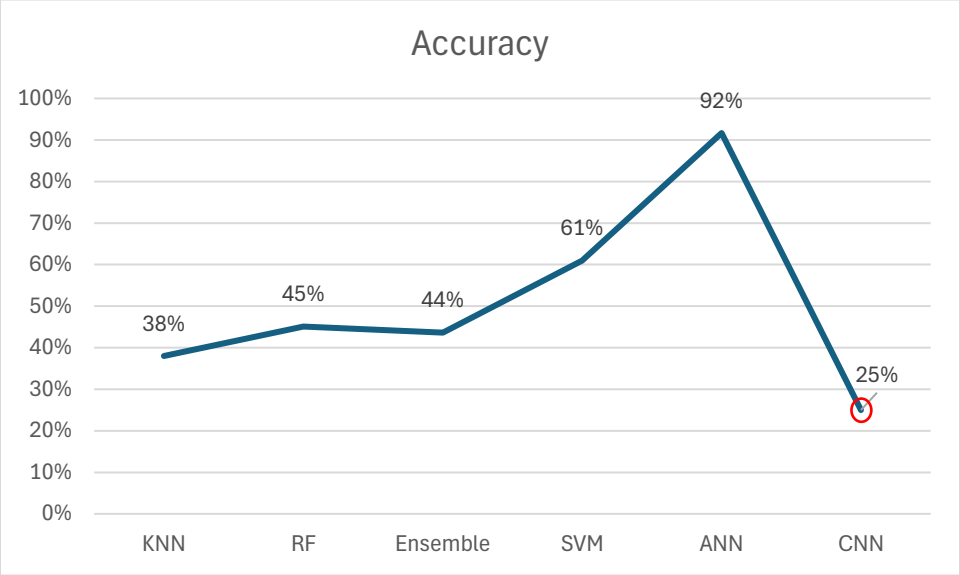


Figure 21. Accuracy Comparison for Models.

10.2 Taguchi Optimization for ANN Model Parameters

Our ANN model consists of a lot of Hyperparameters which rule the accuracy of the results. So, we need to test each combination and evaluate the accuracy of each combination. To consider a good optimization process, Taguchi optimization of the hyperparameter combinations was an appropriate choice. The following parameters were adjusted for every test case.

Table 5. Hyperparameter Combinations for Taguchi Optimization.

Parameter	**Values**
Learning Rate	[0.0001, 0.0003, 0.001]
Batch Size	[16, 32, 64]
Train Splits	[0.7, 0.6, 0.8]
Alpha Values	[0.1, 0.2, 0.3]

Total Number of Test Cases = 81

Here is the table representation of your parameters:

Table 6. Final Parameters for Model Selection.

Parameter	Value
Learning Rate	0.0003
Batch Size	16
Train Split	0.7
Validation Split	0.2
Test Split	0.1
Alpha (Leaky ReLU)	0.1
Test Accuracy	91.67%

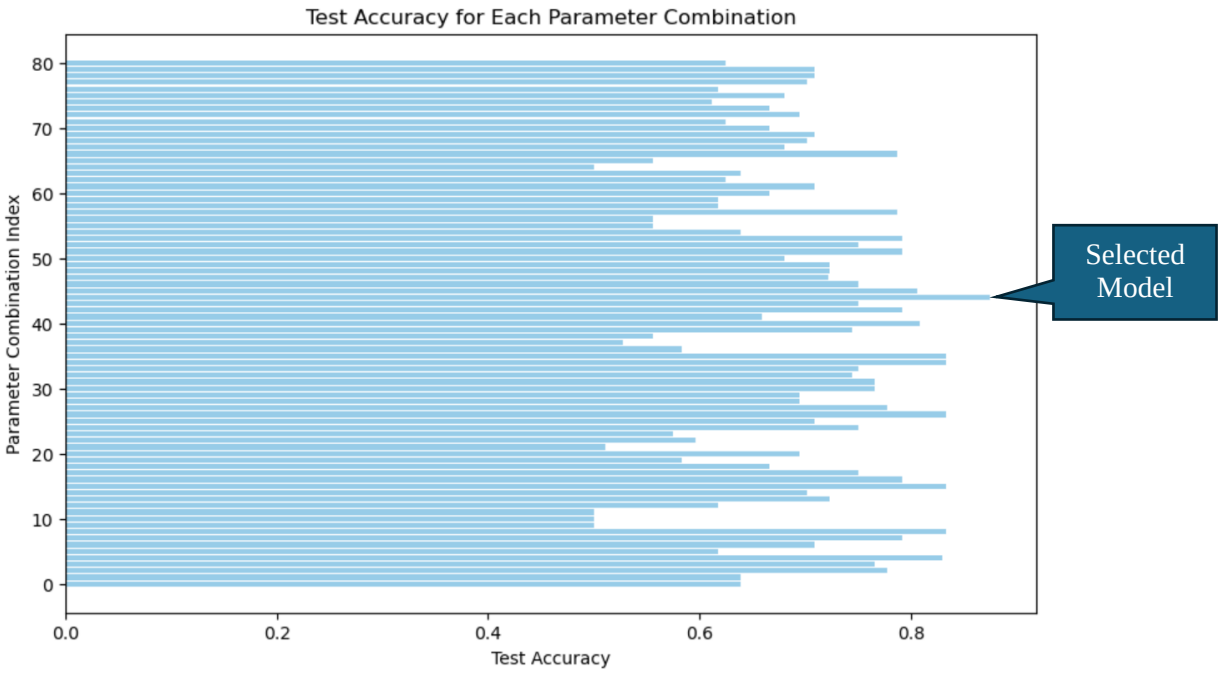


Figure 22. Test Results for Hyperparameter Combinations.

10.3 Effect of the Dataset on Accuracy

The RGB (Red, Green, Blue) values are crucial for encoding color information in data analysis, especially in models including color-based characteristics. Although each color channel adds to the feature set, handling three distinct dimensions for every observation might make the dataset

more difficult, which may have an impact on the effectiveness and precision of machine learning models. Models like k-Nearest Neighbors (k-NN), which are susceptible to the dimensionality of input data because of the curse of dimensionality [30], may be particularly affected by this complexity.

Finding the Euclidean distance of the RGB values for each place is a useful technique for dimensionality reduction. The following is a mathematical expression for the Euclidean distance:

$$\text{Euclidean Distance } = \sqrt{R^2 + G^2 + B^2}$$

The three distinct RGB components are transformed into a single feature via this scalar representation, which captures the color's overall intensity. This kind of dimensionality reduction makes the data easier to handle and less vulnerable to high-dimensional space problems like overfitting and higher processing costs [31]. It has been demonstrated that dimensionality reduction using methods such as calculating the Euclidean distance enhances the performance of distance-based models like k-NN. Simplifying data representations by integrating associated characteristics into a single metric improves classification accuracy and cuts down on calculation time, claim Xu and Wang [32]. Furthermore, this method reduces noise and duplication while preserving the key features of the color data [33].

XI. REAL-TIME APPLICATION

Based on the results above, it can be concluded that we can use this technology for clinical applications to save precious time and remove the room for human error while culturing bacteria manually to detect an infection related to bacteria. In Figure 21 we have an example of a prototype band-aid embedded with the 9 selected dyes on a Calorimetric Sensing array to be used in real-time applications on patients for actual wound monitoring.

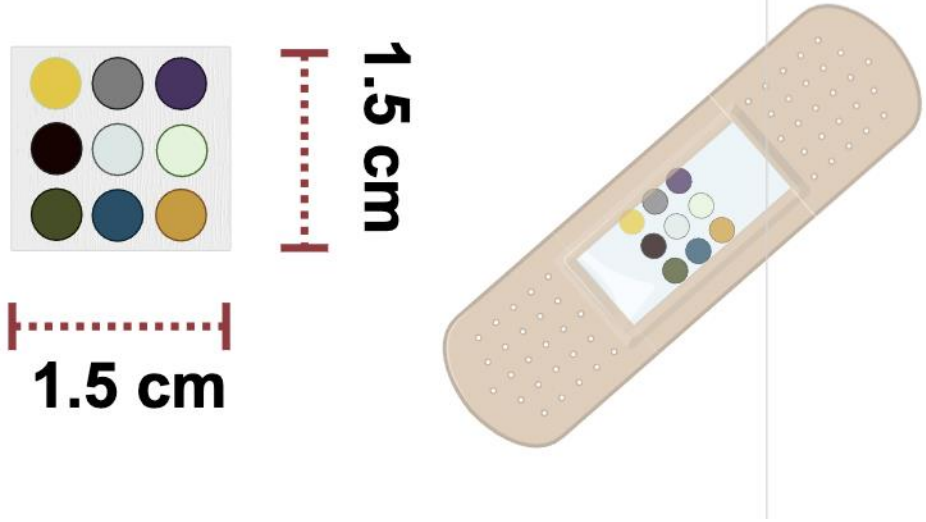


Figure 23. Prototype for the CSA Embedded with Sensor Dyes.

On the next page (Figure are the images for the Front-End we have given to these results so that it can be used by the end consumer for clinical applications.

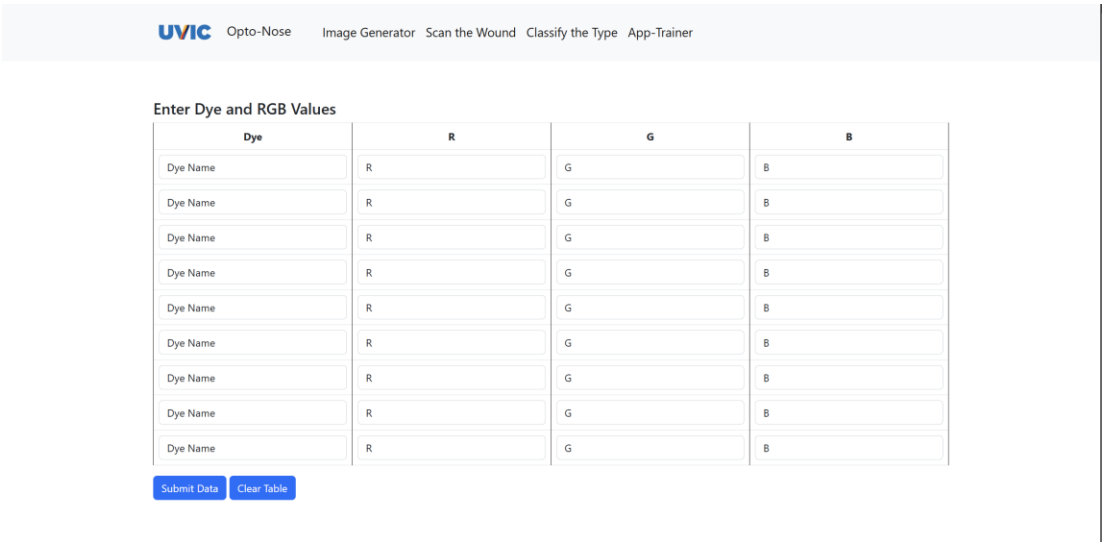


Figure 24. Image Generator from RGB values.

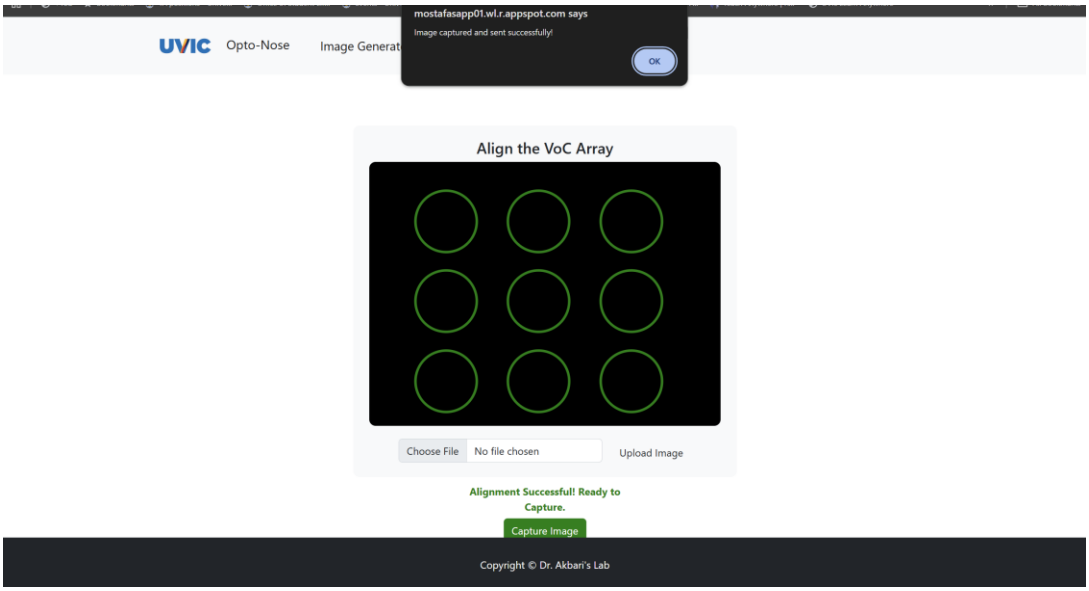


Figure 25. Image Capture or Upload Section for Generating RGB values.

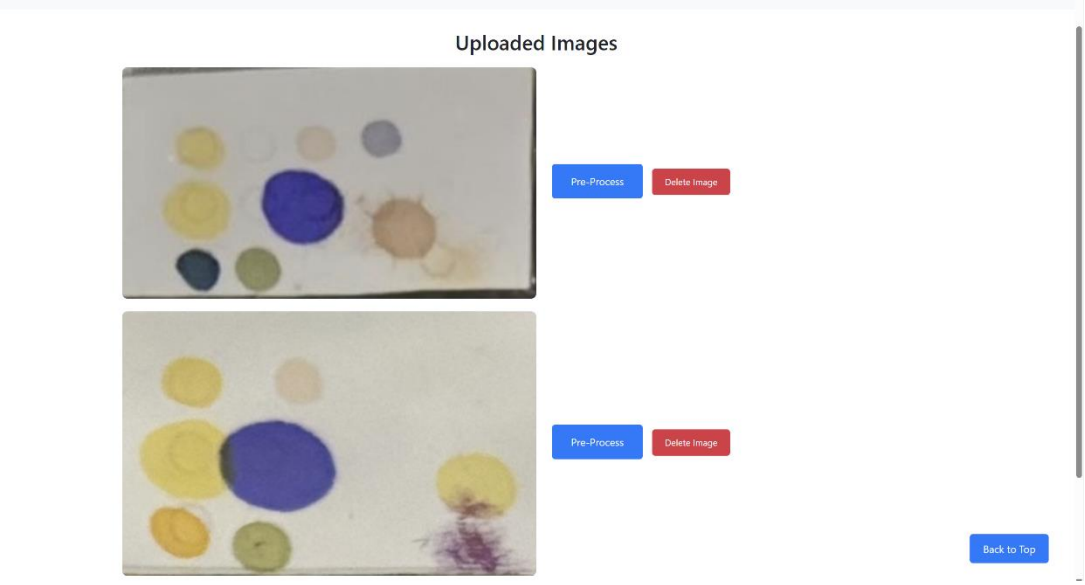


Figure 26. Pre-Processing Section for Captured Images.

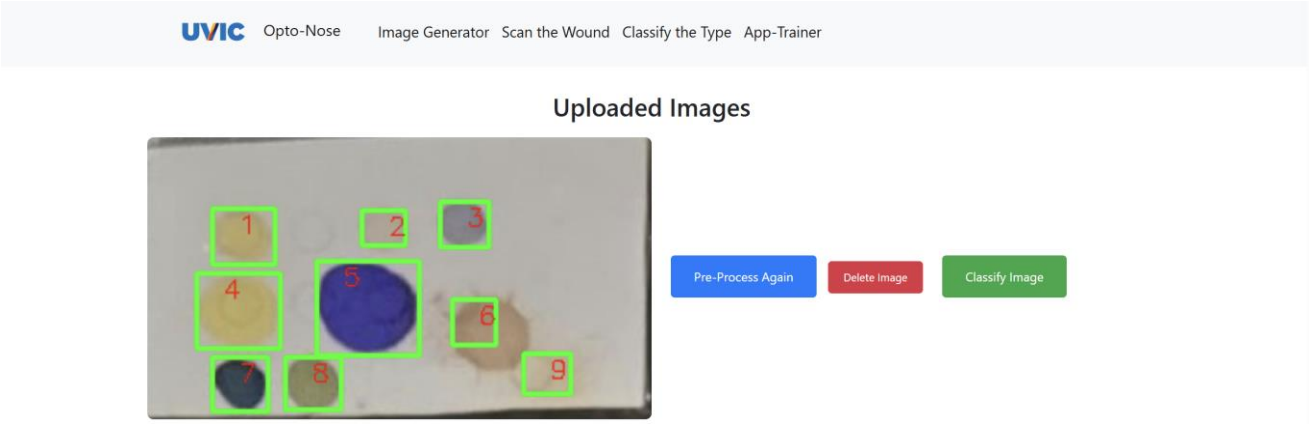


Figure 27. Post-Processing Results (Regions Marked for Sensors)

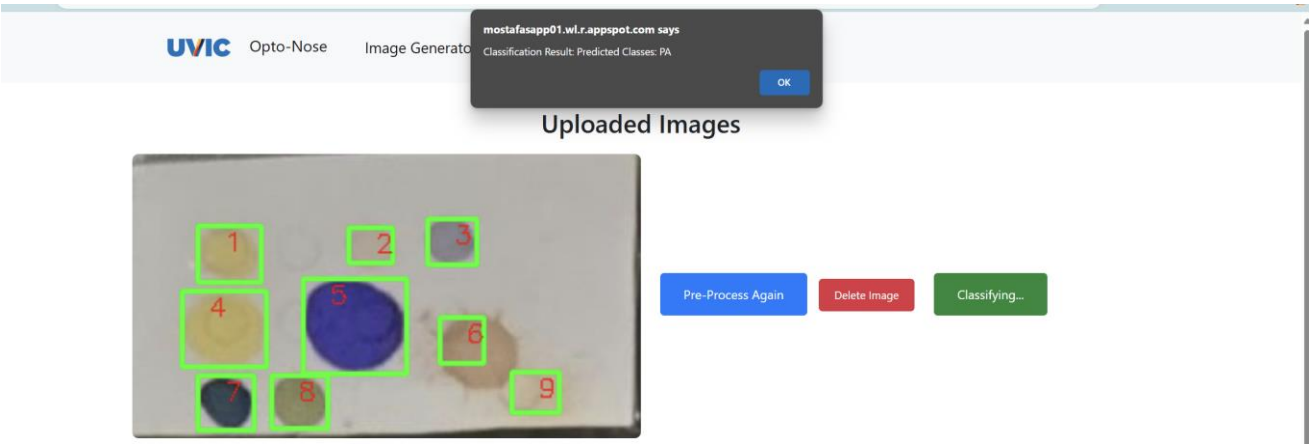


Figure 28. ANN Model Classification Results – Correct Prediction

XII. DISCUSSION

This research represents a notable breakthrough in bacterial identification techniques, providing a flexible and effective approach for detecting various bacterial species across different growth media. The sensor's capacity to precisely identify and distinguish between single and multiple bacterial cultures, irrespective of the culture medium, signifies a substantial advancement in microbial diagnostics. This feature is especially beneficial in medical environments, where swift and accurate pathogen identification is essential for optimal treatment. The unique patterns of color changes observed in the dye when exposed to multi-bacterial cultures create a distinctive signature, enabling more accurate identification of complex microbial populations. This investigation breaks new ground in CSA research, as it is the first to demonstrate the sensor's ability to differentiate between infections caused by single bacteria and those involving multiple bacterial species, across various culture media. This capability has not been explored in previous studies of CSA devices.

The one-hour rapid response time of the sensor could revolutionize bacterial detection. This substantial decrease in detection duration, compared to conventional culture-based or molecular tests that usually take one to two days, has extensive implications for patient care and public health, particularly in wound management. Quicker identification enables more prompt therapeutic interventions, potentially enhancing treatment results, minimizing complication risks, and decreasing wound care expenses. The non-invasive nature of VOC-based sensors has greatly improved bacterial detection techniques. By eliminating the need for bacterial sampling, this approach overcomes the limitations of traditional culture-based and molecular sequencing methods, which can be time-intensive and potentially disrupt the microbial environment. This non-invasive feature not only streamlines the detection process but also allows for continuous monitoring without interfering with the bacterial population being studied.

Furthermore, the sensor design addresses the limitations associated with GC-MS analysis, particularly regarding time efficiency and resource requirements. The absence of VOC capture

and sample preparation steps streamlines the detection process, enhancing its accessibility and practicality for routine use. This simplification may potentially lead to the broader adoption of VOC-based bacterial detection in various fields of medical diagnostics related to bacteria detection. The capacity of the sensor to provide rapid results without requiring specialized equipment or extensive sample handling procedures makes it a promising tool for in situ bacterial detection and monitoring in diverse settings.

The sensor's remarkable stability, storage capacity, and resilience across various environmental conditions, including its ability to withstand changes in temperature and humidity, make it ideal for transport and use in wound diagnostics in healthcare facilities, remote areas, and home settings. The reason for its resilience to environmental disturbances is because of the use of wound skin on the CSA Dyes. The use of wound skin as a dressing is a novel thought by Dr. Mostafa Azimzadeh in the lab and literature works like [44] also helped to validate the choice. The brand that was used for the wound skin was Tegaderm (easily available). A self-adhesive, semi-permeable polyurethane film coating efficiently controls moisture and increases the sensor's resistance to external disruptions such as temperature changes and humidity [44]. The wound skin enables controlled water vapor transmission, avoiding excessive moisture accumulation that could skew sensor data and preserving a dry yet stable microenvironment essential for accurate volatile organic compound (VOC) detection.

Therefore, the VOCs were able to outperform many other VOC sensors, which are highly affected by humidity and lack accuracy. Additionally, when compared to the e-nose developed for detecting bacteria through VOC analysis, this sensor exhibits greater sensitivity and functions without power. These outstanding features enhance its practicality and dependability, positioning this sensor as a valuable tool for scientific research and clinical applications in microbiology and infectious disease management.

The potential of this sensor technology is further enhanced using computational power. Deep Learning makes it possible to analyze intricate VOC patterns and accurately identify bacterial

species by utilizing machine learning techniques. By learning from the distinct signals produced by the sensor, ANN or CNN models can more accurately differentiate between bacterial infections. Furthermore, AI-enabled real-time data processing and predictive analytics guarantee quicker and more precise diagnostic outcomes, especially in crucial contexts like emergency rooms and hospitals. AI and sensor technology work together to produce a potent diagnostic tool that offers previously unheard-of levels of precision, effectiveness, and dependability in the identification of infections.

XIII. FUTURE WORK

This work is in the developing stages and needs various types of tests to prepare it for in-vivo tests on mice models.

13.1 Proposed Testing for the Application (MAK-1)

- The MAK-1 Application is developed on various complex algorithms and requires some vigorous testing for the integrity of results. The following tests can be done to make sure that it can be used at a clinical level.
- RGB Detection Accuracy: Verify RGB value extraction in a range of lighting scenarios and make sure the same input yields consistent results.
- Examine the CNN's accuracy, resilience to changes in input, and ability to make consistent predictions over a series of trials.
- Reproducibility: Make sure that the same results are obtained throughout several trials using the same dataset and on various platforms.
- Performance Under Restrictions: Assess the application's performance under edge circumstances, such as low-quality photos or incomplete RGB circles
- Scalability: Evaluate the effectiveness of the system with bigger datasets or more users at once.
- Error Handling: Evaluate the program's capacity to recognize and fix mistakes or invalid inputs. Verify results experimentally by comparing them to ground truth data obtained using accepted laboratory techniques.
- Usability and Stability: Examine for long-term dependability under clinical workloads and an intuitive user experience.

13.2 Improving the CNN Model Accuracy

Expanding the quantity of the training dataset is essential to raising the CNN model's accuracy. Even with roughly 500 high-quality and diverse images per class, CNNs can still perform competitively in many classification tasks when combined with data augmentation approaches, even though deep learning models like CNNs are known to benefit from huge datasets [30]. For example, by adding transformations like rotation, scaling, and cropping, data augmentation can efficiently increase the size of small samples, enhancing model generalization [31], [39]. Additionally, the application will have the ability to train the model directly, allowing for ongoing learning as fresh photos are gathered. To fulfill the demands of real-world settings, this iterative technique guarantees that the model develops and improves dynamically [32].

In our case, the accuracy results of 25% were below the needed accuracy from similar techniques (>90%). However, we have other literature to back our hypothesis to increase the accuracy to at least 95%. The image library has 235 high-quality phones as of now which is one of the reasons for the poor performance. As more and more trials are conducted and data is generated, those images provide rich information to enhance the performance of the model and produce better results.

XIV. BIBLIOGRAPHY

[1] N. Yusuf et al., "In-vitro diagnosis of single and poly microbial species targeted for diabetic foot infection using e-nose technology," BMC Bioinformatics, vol. 16, no. 158, 2015. DOI: 10.1186/s12859-015-0601-5.

[2] Z. Li, J. R. Askim, and K. S. Suslick, "The optoelectronic nose: Colorimetric and fluorometric sensor arrays," Chemical Reviews, vol. 119, no. 231, pp. 231–292, 2019. DOI: 10.1021/acs.chemrev.8b00226.

[3] J. R. Carey et al., "Rapid identification of bacteria with a disposable colorimetric sensing array," Journal of the American Chemical Society, vol. 133, no. 7571, pp. 7571–7576, 2011. DOI: 10.1021/ja201634d.

[4] G. Niedermayer, “Investigations of calorimeter clustering in Atlas using machine learning,” M.S. thesis, Dept. Phys. Astron., Univ. Victoria, Victoria, BC, Canada, 2017.

[5] X.-Y. Gong, H. Su, D. Xu, Z.-T. Zhang, F. Shen, and H.-B. Yang, “An overview of contour detection approaches,” Int. J. Autom. Comput., vol. 15, no. 3, pp. 321–335, Jun. 2018. doi: 10.1007/s11633-018-1117-z.

[6] ATCC, “Bacteriology Culture Guide,” version BCG-122021-v09, 2022. Available: www.atcc.org.

[7] M. T. Madigan, K. S. Bender, D. H. Buckley, W. M. Sattley, and D. A. Stahl, Brock Biology of Microorganisms, 15th ed. Boston, MA: Pearson, 2018.

[8] Z. Li, J. R. Askim, and K. S. Suslick, “The optoelectronic nose: Colorimetric and fluorometric sensor arrays,” Chem. Rev., vol. 119, no. 1, pp. 231–292, 2019. doi: 10.1021/acs.chemrev.8b00226.

[9] H. C. Nguyen and A. B. Holmes, “Standards for CFU determination in bacterial studies,” Microb. Methods J., vol. 72, pp. 101–108, 2017. doi: 10.1016/j.mim.2017.04.001.

[10] J. T. Lee et al., “Standardization of bacterial concentration using spectrophotometry,” J. Microbiol. Tech., vol. 45, pp. 254–259, 2021. doi: 10.1016/j.jmicro.2020.12.015.

[11] R. Gupta et al., “Optimizing bacterial culture conditions for VOC analysis,” Anal. Biochem., vol. 540, pp. 33–40, 2019. doi: 10.1016/j.ab.2018.10.007.

[12] P. Patel and J. Smith, “VOC interactions in mixed microbial cultures,” J. Med. Microbiol., vol. 69, no. 7, pp. 928–935, 2020. doi: 10.1099/jmm.0.001254.

[13] C. Liu and M. Chen, “Impact of growth media on bacterial VOC production,” Biotech Adv., vol. 38, pp. 107–116, 2019. doi: 10.1016/j.biotechadv.2019.01.006.

[14] S. Brown et al., “Sensor array design for optimal VOC capture,” IEEE Sens. J., vol. 19, no. 11, pp. 4214–4223, 2019. doi: 10.1109/JSEN.2019.2901043.

[15] Q. Liu, Y. Wang, and H. Chen, “Deep learning approaches for bacterial classification using CNNs,” IEEE Trans. Biomed. Eng., vol. 69, no. 3, pp. 685–693, 2022. doi: 10.1109/TBME.2022.3152345.

[16] X. Chen et al., “Portable analytical techniques for monitoring volatile organic chemicals in biomanufacturing processes: recent advances and limitations,” Front. Chem., vol. 8, p. 837, 2020. doi: 10.3389/fchem.2020.00837.

[17] Olivares, A., Dryahina, K., Navarro, J. L., Smith, D., Spaněl, P., & Flores, M. (2011). SPME-GC-MS versus Selected Ion Flow Tube Mass Spectrometry (SIFT-MS) analyses for the study of volatile compound generation and oxidation status during dry fermented sausage processing. *Journal of agricultural and food chemistry*, *59*(5), 1931–1938. https://doi.org/10.1021/jf104281a

[18] C. Liu et al., “Impact of growth media on bacterial VOC production,” Biotechnol. Adv., vol. 38, pp. 107–116, 2019. doi: 10.1016/j.biotechadv.2019.01.006.

[19] J. Guarrasi et al., “The individual contribution of starter and non-starter lactic acid bacteria to VOC composition in cheese,” Int. J. Food Microbiol., vol. 259, pp. 35–42, 2017. doi: 10.1016/j.ijfoodmicro.2017.07.022.

[20] H. J. Ko and T. H. Park, “Bioelectronic nose and its application to smell visualization,” J. Biol. Eng., vol. 10, p. 17, 2016. doi: 10.1186/s13036-016-0041-4.

[21] J. R. Askim et al., “Optical sensor arrays for chemical sensing: the optoelectronic nose,” Chem. Soc. Rev., vol. 42, pp. 8649–8682, 2013. doi: 10.1039/c3cs60179j.

[22] N. Yusuf et al., “In-vitro diagnosis of single and poly microbial species using e-nose technology,” BMC Bioinformatics, vol. 16, no. 158, 2015. doi: 10.1186/s12859-015-0601-5.

[23] Z. Li et al., “The optoelectronic nose: colorimetric and fluorometric sensor arrays,” Acc. Chem. Res., vol. 54, pp. 950–960, 2021. doi: 10.1021/acs.accounts.0c00671.

[24] L. Jones et al., “Detection of polymicrobial VOC profiles in clinical samples,” Microbial Pathogenesis, vol.142, p. 104026, 2020. doi: 10.1016/j.micpath.2020.104026.

[25] R. Vishinkin and H. Haick, “Nanoscale sensor technologies for disease detection via volatolomics,” Small, vol. 11, no. 46, pp. 6142–6164, 2015. doi: 10.1002/smll.201501904.

[26] M. T. Madigan et al., Brock Biology of Microorganisms, 15th ed. Boston, MA: Pearson, 2018.

[27] Calderon-Santoyo et al., “Monitoring lactic fermentation using gas chromatography and E-nose,” Engineering, vol. 5, pp. 13–19, 2013. doi: 10.4236/eng.2013.59A002.

[28] R. Gupta et al., “Optimizing bacterial culture conditions for VOC analysis,” Anal. Biochem., vol. 540, pp. 33–40, 2019. doi: 10.1016/j.ab.2018.10.007.

[29]Bellman, R. (1961). Adaptive Control Processes: A Guided Tour. Princeton University Press.

[30]Bishop, C. M. (2006). Pattern Recognition and Machine Learning. Springer.

[31]Xu, Y., & Wang, Y. (2017). Feature Engineering for Machine Learning: Principles and Techniques. Cambridge University Press.

[32]Duda, R. O., Hart, P. E., & Stork, D. G. (2001). Pattern Classification (2nd ed.). Wiley-Interscience.

[33]Guyon, I., & Elisseeff, A. (2003). An introduction to variable and feature selection. Journal of Machine Learning Research, 3, 1157–1182.

[34] Jain, A. K., & Duin, R. P. W. (2004). Statistical pattern recognition: A review. IEEE Transactions on Pattern Analysis and Machine Intelligence, 22(1), 4–37.

[35] Zha, C., Li, L., Zhu, F., & Zhao, Y. (2024). The Classification of VOCs Based on Sensor Images Using a Lightweight Neural Network for Lung Cancer Diagnosis. Sensors, 24(2818). https://doi.org/10.3390/s24092818&#8203.

[36] Qu, X., Hu, Y., Xu, C., Li, Y., Zhang, L., Huang, Q., Moshirian-Farahi, S. S., Zhang, J., Xu, X., Liao, M., & Fu, Y. (2024). Optical sensors of volatile organic compounds for non-invasive diagnosis of diseases. *Chemical Engineering Journal*, *485*, 149804. https://doi.org/10.1016/j.cej.2024.149804

[37] Rolnick, D., Veit, A., Belongie, S., & Shavit, N. "Deep learning is robust to massive label noise." *Advances in Neural Information Processing Systems*, vol. 30, pp. 1–12, 2017.

[38] Shorten, T., & Khoshgoftaar, T. M. "A survey on image data augmentation for deep learning." Journal of Big Data, vol. 6, no. 1, pp. 1–48, 2019.

[39] Brownlee, J. Better Deep Learning: Train Faster, Reduce Overfitting, and Make Better Predictions. Machine Learning Mastery, 2018.

[40] K. Sathupadi, R. Avula, A. Velayutham, and S. Achar, “RAP-Optimizer: Resource-Aware Predictive Model for Cost Optimization of Cloud AIaaS Applications,” *Preprints*, Oct. 8, 2024, doi: 10.20944/preprints202410.0504.v1.

[41] S. Riedel and K. C. Carroll, “Current clinical procedures for bacterial infections,” *J. Infect. Chemother.*, vol. 16, no. 4, pp. 301–306, 2010.

[42] S. Schulz and J. S. Dickschat, “Bacterial volatiles: The smell of small organisms,” *Nat. Prod. Rep.*, vol. 24, no. 4, pp. 814–842, 2007.

[43] M. Kai, M. Haustein, F. Molina, A. Petri, B. Scholz, and B. Piechulla, “Bacterial volatiles and their action potential,” *Appl. Microbiol. Biotechnol.*, vol. 81, no. 6, pp. 1001–1012, 2009.

[44] A. D. Russell, J. R. Hugo, and I. W. Hancock, “Use of polyurethane dressings to reduce microbial contamination,” *J. Clin. Microbiol.*, vol. 38, no. 4, pp. 1620–1624, 2000.

[45] Azimzadeh, M., Akbari, M., Hoorfar, M., et al. 2024. Bacteria Identification Using Colorimetric Sensor Array for Volatile Organic Compounds (VOC) in Wound Monitoring.

XV. APPENDIX

The code for the MAK-1 App is give below:

Front End: Home.html

```
<!DOCTYPE html>

<html lang="en">

  <head>

    <meta charset="utf-8" />

    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

    <meta name="description" content="" />

    <meta name="author" content="KVS" />

    <title>Mostafa's Application</title>

    <!-- Favicon-->

    <link rel="icon" type="image/png" href="{{ url_for('static', filename='images/Martlet-blue.png') }}" />

    <!-- Bootstrap icons-->

    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css"
rel="stylesheet" />

    <!-- Core theme CSS (includes Bootstrap)-->

    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">

    <style>

      /* Adjust Navbar Height */

      .navbar {

        padding-top: 20px;

        padding-bottom: 20px;

        font-size: 1.2rem;

      }



      /* Fixed position for flash messages with glow */
```

```
.flashed-messages {
    position: fixed;
    top: 80px;
    left: 50%;
    transform: translateX(-50%);
    z-index: 1000;
    text-align: center;
    width: 60%;
}

/* Success message with green glow */
.alert-success {
    box-shadow: 0 0 15px 3px rgba(0, 255, 0, 0.6) !important;
    border-radius: 10px;
    color: green !important;
    background-color: white !important;
    border: 1px solid green;
}

/* Danger (error) message with red glow */
.alert-danger {
    box-shadow: 0 0 15px 3px rgba(255, 0, 0, 0.6) !important;
    border-radius: 10px;
    color: red !important;
    background-color: white !important;
    border: 1px solid red;
}
```

```
/* Content should push the footer down */
html, body {
  height: 100%;
  margin: 0;
  display: flex;
  flex-direction: column;
}

.content {
  flex: 1 0 auto;
  padding: 20px 0;
}

.file-upload-section {
  margin-bottom: 30px;
  text-align: center; /* Centering the upload form */
  background-color: #f8f9fa;
  padding: 20px;
  border-radius: 10px;
}

/* Reduce the space between navbar and main content */
main.content {
  padding-top: 10px;
}

section.py-5 {
  padding-top: 0 !important;
```

```
}

/* Style for the generated image */
.image-section {
  margin-top: 20px;
}
/* Styles for table */
table {
  width: 100%;
  border-collapse: collapse;
  margin-bottom: 20px;
}
table, th, td {
  border: 1px solid black; /* Adds black border around all table cells */
}
th, td {
  padding: 10px;
  text-align: center;
}

th {
  background-color: #f2f2f2;
  text-align: center; /* Center headers */
}

td {
  contenteditable: true; /* Make table cells editable for easy paste */
}
```

```
.table-section {
    margin-top: 30px;
}

.table-submit-btn {
    margin-top: 20px;
}
/* Button to scroll back to home */
.scroll-to-top {
    position: fixed;
    bottom: 40px;
    right: 40px;
    display: none;
    background-color: #007bff;
    color: white;
    padding: 10px 20px;
    border-radius: 5px;
    text-align: center;
    cursor: pointer;
    z-index: 999;
}

.scroll-to-top:hover {
    background-color: #0056b3;
}

/* Smooth scrolling behavior */
```

```
        html {
          scroll-behavior: smooth;
        }
      </style>
    </head>
    <body>
      <!-- Navigation-->
      <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container px-4 px-lg-5">
          <a class="navbar-brand" href="{{ url_for('home')}}">
            <img src="https://storage.googleapis.com/mostafasapp01.appspot.com/static/images/UVic-mark.png" alt="UVic Logo" style="height: 40px; margin-right: 10px;">
            Opto-Nose
          </a>
          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0 ms-lg-4">
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('rgbarray')}}">Image Generator</a></li>
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('capture_image') }}">Scan the Wound</a></li>
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('image_library') }}">Classify the Type</a></li>
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="#!">App-Trainer</a></li>
            </ul>
          </div>
```

```
    </div>
  </nav>
  <main class="content">
    <section class="container py-5">
      <h2 class="text-center">Optoelectronic Noses and Machine Learning for Bacteria Classification</h2>

      <div class="mt-4">
        <h3>What is an Optoelectronic Nose?</h3>
        <p>
          Optoelectronic noses, also known as electronic noses or "e-noses," are advanced sensor systems that mimic the olfactory system to detect and identify various chemical compounds in the air. They consist of an array of sensors that react to specific volatile organic compounds (VOCs), producing a unique response pattern for each compound or mixture. These sensors typically operate based on changes in optical, electrical, or chemical properties when exposed to certain chemicals, with optoelectronic noses focusing on optical signals.
        </p>

        <h4>Working Principle</h4>
        <ul>
          <li><strong>Sensor Array:</strong> An optoelectronic nose comprises a sensor array where each sensor is sensitive to a particular range of VOCs. Changes in properties upon interaction with chemicals produce a measurable signal.</li>
          <li><strong>Data Collection:</strong> The sensor responses are captured as colorimetric or spectral data, often represented as RGB values for each sensor in the array.</li>
          <li><strong>Pattern Recognition:</strong> Optoelectronic noses use pattern recognition algorithms to interpret these complex signals, distinguishing between different VOCs or mixtures.</li>
        </ul>

        <h4>Applications</h4>
        <p>Optoelectronic noses are utilized in various fields, including:</p>
        <ul>
          <li>Medical diagnostics (detecting infections through breath analysis)</li>
          <li>Environmental monitoring (pollution detection)</li>
```

```
            <li>Food quality control (detecting spoilage)</li>

            <li>Security (detecting explosives or illegal substances)</li>

          </ul>

        </div>



        <div class="mt-4">

          <h3>Using Machine Learning for Bacteria Classification</h3>

          <p>
```

For this project, machine learning (ML) is employed to classify bacterial species based on unique response patterns captured by the optoelectronic nose sensor array. Here's why ML is essential in this context:

```
          </p>

          <ul>

            <li><strong>Handling High-Dimensional Data:</strong> The sensor array produces complex data, and ML algorithms are capable of identifying intricate patterns and correlations that might not be apparent otherwise.</li>

            <li><strong>Enhanced Classification Accuracy:</strong> By learning from labeled data, ML models can distinguish between subtle differences in response patterns, resulting in higher classification accuracy.</li>

            <li><strong>Adaptability and Scalability:</strong> ML models can be improved over time with new data, making the system scalable to detect more bacteria species in the future.</li>

            <li><strong>Automated, Real-Time Analysis:</strong> ML enables real-time processing of sensor data, providing immediate feedback, which is crucial in medical or environmental applications where timely information is needed.</li>

            <li><strong>Robustness Against Variability:</strong> ML algorithms can learn to recognize patterns despite environmental variations, enhancing the reliability of the classification system.</li>

          </ul>

        </div>



        <div class="mt-4">

          <h3>Societal Impact and Benefits</h3>

          <p>
```

Implementing machine learning to classify bacteria species using an optoelectronic nose has significant societal benefits, particularly in healthcare, public health, and environmental monitoring. Here's

```
how:

            </p>

            <ul>

              <li><strong>Early Detection and Diagnosis of Infections:</strong> Quick identification of infections enables faster treatment and helps prevent the spread of diseases.</li>

              <li><strong>Reducing Antibiotic Misuse:</strong> Accurate bacteria identification allows for targeted antibiotic use, helping combat antibiotic resistance.</li>

              <li><strong>Non-Invasive, Rapid Testing:</strong> A non-invasive diagnostic alternative offers faster testing without the discomfort of traditional methods.</li>

              <li><strong>Environmental and Food Safety Monitoring:</strong> The technology can detect bacteria in the environment or food products, ensuring public safety.</li>

              <li><strong>Portable, Cost-Effective Solution:</strong> Compared to conventional lab equipment, optoelectronic noses are more affordable and portable, ideal for resource-limited settings.</li>

              <li><strong>Rapid Response to Public Health Outbreaks:</strong> Quick identification of bacterial pathogens in outbreaks can aid in containment efforts.</li>

              <li><strong>Reducing Burden on Clinical Laboratories:</strong> Automated ML-based classification reduces the workload of manual testing, improving efficiency in healthcare.</li>

            </ul>

          </div>

        </section>

      </main>

      <!-- Back to Home Button -->

      <div class="scroll-to-top" id="scrollTopBtn">

        Back to Home

      </div>



      <!-- Footer-->

      <footer class="py-5 bg-dark">

        <div class="container">

          <p class="m-0 text-center text-white">Copyright &copy; Dr. Akbari's Lab</p>

        </div>
```

```
</footer>


<!-- Bootstrap core JS-->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
<!-- Core theme JS-->
<script src="{{ url_for('static', filename='js/scripts.js') }}"></script>
<script>
  // Automatically scroll to the generated image when it is visible
  window.onload = function() {
    if (document.querySelector('#generated-image-section img')) {
      document.querySelector('#generated-image-section').scrollIntoView({ behavior: 'smooth' });
    }
  };

  // Back to Home Button
  const scrollTopBtn = document.getElementById('scrollTopBtn');

  window.onscroll = function() {
    if (document.body.scrollTop > 100 || document.documentElement.scrollTop > 100) {
      scrollTopBtn.style.display = "block";
    } else {
      scrollTopBtn.style.display = "none";
    }
  };

  // Scroll back to the top when the button is clicked
  scrollTopBtn.addEventListener('click', function() {
    window.scrollTo({ top: 0, behavior: 'smooth' });
```

```
        });
      </script>
    </body>
</html>
```

Front End: Image Table Section

```
<!DOCTYPE html>
<html lang="en">
  <head>
    <meta charset="utf-8" />
    <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
    <meta name="description" content="" />
    <meta name="author" content="KVS" />
    <title>Mostafa's Application</title>
    <!-- Favicon-->
    <link rel="icon" type="image/png" href="{{ url_for('static', filename='images/Martlet-blue.png') }}" />
    <!-- Bootstrap icons-->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css"
rel="stylesheet" />
    <!-- Core theme CSS (includes Bootstrap)-->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
    <style>
      /* Adjust Navbar Height */
      .navbar {
        padding-top: 20px;
        padding-bottom: 20px;
        font-size: 1.2rem;
      }
```

```
/* Fixed position for flash messages with glow */
.flashed-messages {
    position: fixed;
    top: 80px;
    left: 50%;
    transform: translateX(-50%);
    z-index: 1000;
    text-align: center;
    width: 60%;
}

/* Success message with green glow */
.alert-success {
    box-shadow: 0 0 15px 3px rgba(0, 255, 0, 0.6) !important;
    border-radius: 10px;
    color: green !important;
    background-color: white !important;
    border: 1px solid green;
}

/* Danger (error) message with red glow */
.alert-danger {
    box-shadow: 0 0 15px 3px rgba(255, 0, 0, 0.6) !important;
    border-radius: 10px;
    color: red !important;
    background-color: white !important;
    border: 1px solid red;
}
```

```
/* Content should push the footer down */
html, body {
  height: 100%;
  margin: 0;
  display: flex;
  flex-direction: column;
}

.content {
  flex: 1 0 auto;
  padding: 20px 0;
}

.file-upload-section {
  margin-bottom: 30px;
  text-align: center; /* Centering the upload form */
  background-color: #f8f9fa;
  padding: 20px;
  border-radius: 10px;
}

/* Reduce the space between navbar and main content */
main.content {
  padding-top: 10px;
}

section.py-5 {
```

```
    padding-top: 0 !important;
}

/* Style for the generated image */
.image-section {
    margin-top: 20px;
}
/* Styles for table */
table {
    width: 100%;
    border-collapse: collapse;
    margin-bottom: 20px;
}
table, th, td {
    border: 1px solid black; /* Adds black border around all table cells */
}
th, td {
    padding: 10px;
    text-align: center;
}

th {
    background-color: #f2f2f2;
    text-align: center; /* Center headers */
}

td {
    contenteditable: true; /* Make table cells editable for easy paste */
```

```
}

.table-section {
    margin-top: 30px;
}

.table-submit-btn {
    margin-top: 20px;
}
/* Button to scroll back to home */
.scroll-to-top {
    position: fixed;
    bottom: 40px;
    right: 40px;
    display: none;
    background-color: #007bff;
    color: white;
    padding: 10px 20px;
    border-radius: 5px;
    text-align: center;
    cursor: pointer;
    z-index: 999;
}

.scroll-to-top:hover {
    background-color: #0056b3;
}
```

```
        /* Smooth scrolling behavior */
        html {
          scroll-behavior: smooth;
        }
      </style>
    </head>
    <body>
      <!-- Navigation-->
      <nav class="navbar navbar-expand-lg navbar-light bg-light">
        <div class="container px-4 px-lg-5">
          <a class="navbar-brand" href="{{ url_for('home')}}">
            <img src="https://storage.googleapis.com/mostafasapp01.appspot.com/static/images/UVic-mark.png" alt="UVic Logo" style="height: 40px; margin-right: 10px;">
            Opto-Nose
          </a>
          <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
            <span class="navbar-toggler-icon"></span>
          </button>
          <div class="collapse navbar-collapse" id="navbarSupportedContent">
            <ul class="navbar-nav me-auto mb-2 mb-lg-0 ms-lg-4">
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('rgbarray')}}">Image Generator</a></li>
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('capture_image') }}">Scan the Wound</a></li>
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('image_library') }}">Classify the Type</a></li>
              <li class="nav-item"><a class="nav-link active" aria-current="page" href="#!">App-Trainer</a></li>
            </ul>
```

```
        </div>
      </div>
    </nav>

    <!-- Main Content Section-->
    <main class="content">
      <section class="py-5">
        <div class="container px-4 px-lg-5 my-5">
          <!-- Flashed Messages Section -->
          <div class="flashed-messages">
            <!-- Handle flashed messages -->
            {% with messages = get_flashed_messages(with_categories=True) %}
              {% if messages %}
                <div>
                  {% for category, message in messages %}
                    <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">
                      {{ message }}
                      <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>
                    </div>
                  {% endfor %}
                </div>
              {% endif %}
            {% endwith %}
          </div>
          <!-- Upload Button Section
          <div class="row gx-4 gx-lg-5 align-items-center">
            <div class="col-md-6 d-flex justify-content-center align-items-center">
```

```
            <div class="file-upload-section mb-5">
              <h5>Upload the RGB File</h5>
              <form method="POST" enctype="multipart/form-data" action="/upload">
                <input type="file" class="form-control mb-3" name="file" id="fileInput" required>
                <button class="btn btn-primary" type="submit">Upload File</button>
              </form>
            </div>
          </div> -->
        <!-- Table Section -->
        <div class="table-section">
          <h4>Enter Dye and RGB Values</h4>
          <form id="rgbForm" method="POST" action = "{{ url_for('process_table') }}">
            <table class="table table-bordered">
              <thead>
                <tr>
                  <th class="text-center">Dye</th>
                  <th class="text-center">R</th>
                  <th class="text-center">G</th>
                  <th class="text-center">B</th>
                </tr>
              </thead>
              <tbody>
                {% for i in range(9) %}
                <tr>
                  <td>
                    <input type="text" class="form-control" name="dye_name_{{ i }}" value="{{ dyes[i] if dyes
else 'Dye Name' }}">
                  </td>
```

```
                        <td>
                            <input type="text" class="form-control" name="r_{{ i }}" value="{{ rgb_values[i][0] if
rgb_values else 'R' }}">
                        </td>
                        <td>
                            <input type="text" class="form-control" name="g_{{ i }}" value="{{ rgb_values[i][1] if
rgb_values else 'G' }}">
                        </td>
                        <td>
                            <input type="text" class="form-control" name="b_{{ i }}" value="{{ rgb_values[i][2] if
rgb_values else 'B' }}">
                        </td>
                    </tr>
                    {% endfor %}
                </tbody>
            </table>
            <button type="submit" class="btn btn-primary">Submit Data</button>
            <button type="button" class="btn btn-primary" onclick="clearTable()">Clear Table</button>
        </form>
        </div>
        <!-- Generated Image Section -->
        <div id = "generated-image-section" class="col-md-6 image-section">
          {% if image_path %}
          <h5>Generated Image:</h5>
          <img src="{{ image_path }}" alt="Generated Image" class="img-fluid" />
          {% endif %}
        </div>
        </div>
    </div>
```

```
  </section>
</main>

<!-- Back to Home Button -->
<div class="scroll-to-top" id="scrollTopBtn">
  Back to Top
</div>

<!-- Footer-->
<footer class="py-5 bg-dark">
  <div class="container">
    <p class="m-0 text-center text-white">Copyright &copy; Dr. Akbari's Lab</p>
  </div>
</footer>

<!-- Bootstrap core JS-->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
<!-- Core theme JS-->
<script src="{{ url_for('static', filename='js/scripts.js') }}"></script>
<script>
  // Automatically scroll to the generated image when it is visible
  window.onload = function() {
    if (document.querySelector('#generated-image-section img')) {
      document.querySelector('#generated-image-section').scrollIntoView({ behavior: 'smooth' });
    }
  };

  // Back to Home Button
```

```
    const scrollTopBtn = document.getElementById('scrollTopBtn');

    window.onscroll = function() {
      if (document.body.scrollTop > 100 || document.documentElement.scrollTop > 100) {
        scrollTopBtn.style.display = "block";
      } else {
        scrollTopBtn.style.display = "none";
      }
    };

    // Scroll back to the top when the button is clicked
    scrollTopBtn.addEventListener('click', function() {
      window.scrollTo({ top: 0, behavior: 'smooth' });
    });
  </script>
  <script>
    document.addEventListener('DOMContentLoaded', function() {
      const table = document.querySelector('#rgbForm');

      table.addEventListener('paste', function(e) {
        const data = e.clipboardData.getData('text');
        const rows = data.split('\n');

        const cells = table.querySelectorAll('input');
        let currentCell = 0;

        rows.forEach(row => {
          const values = row.split(/\t|\s+/).filter(value => value); // Split by tabs or spaces
```

```
            values.forEach(value => {
              if (cells[currentCell]) {
                cells[currentCell].value = value.trim();
                currentCell++;
              }
            });
          });

          e.preventDefault(); // Prevent default paste behavior
        });
      });
    </script>
    <script>
    function clearTable() {
      // Get all input fields in the form and clear their values
      var inputs = document.querySelectorAll('#rgbForm input');
      inputs.forEach(function(input) {
        input.value = '';
      });
    }
    </script>
  </body>
</html>
```

Front End: Image Capture Section

```
<!DOCTYPE html>
<html lang="en">
<head>
```

```
  <meta charset="utf-8" />

  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />

  <title>Mostafa's Application</title>

  <link rel="icon" type="image/png" href="{{ url_for('static', filename='images/Martlet-blue.png') }}" />

  <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" rel="stylesheet"
/>

  <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">

  <style>

/* Fixed position for flash messages with glow */

    .flashed-messages {

      position: fixed;

      top: 80px;

      left: 50%;

      transform: translateX(-50%);

      z-index: 1000;

      text-align: center;

      width: 60%;

    }



    /* Success message with green glow */

    .alert-success {

      box-shadow: 0 0 15px 3px rgba(0, 255, 0, 0.6) !important;

      border-radius: 10px;

      color: green !important;

      background-color: white !important;

      border: 1px solid green;

    }
```

```
/* Danger (error) message with red glow */
.alert-danger {
  box-shadow: 0 0 15px 3px rgba(255, 0, 0, 0.6) !important;
  border-radius: 10px;
  color: red !important;
  background-color: white !important;
  border: 1px solid red;
}
.navbar {
  padding-top: 20px;
  padding-bottom: 20px;
  font-size: 1.2rem;
}
.image-capture-section {
  position: relative;
  text-align: center;
  background-color: #f8f9fa;
  padding: 20px;
  border-radius: 10px;
  max-width: 600px;
  margin: 0 auto;
}
.circle-grid {
  position: absolute;
  top: 50%;
  left: 50%;
  transform: translate(-50%, -50%);
  display: grid;
```

```
    grid-template-columns: repeat(3, 120px);
    grid-gap: 25px;
    justify-content: center;
    pointer-events: none;
}
.circle {
    width: 100px;
    height: 100px;
    border-radius: 50%;
    background-color: transparent;
    border: 4px solid red;
    transition: border-color 0.3s ease;
}
.circle.aligned {
    border-color: green;
}
video, canvas {
    display: block;
    margin: 0 auto;
    width: 100%;
    max-width: 550px;
    border-radius: 10px;
}
#captureButton {
    position: absolute;
    top: 110%;
    left: 50%;
    transform: translateX(-50%);
```

```
        margin-top: 10px;
        background-color: green;
        color: white;
      }
      .flash-message {
        display: none;
        position: absolute;
        top: 100%;
        left: 50%;
        transform: translateX(-50%);
        color: green;
        font-weight: bold;
        margin-top: 10px;
      }
    </style>
</head>
<body>
    <!-- Navigation-->
    <nav class="navbar navbar-expand-lg navbar-light bg-light">
      <div class="container px-4 px-lg-5">
        <a class="navbar-brand" href="{{ url_for('home')}}">
          <img src="https://storage.googleapis.com/mostafasapp01.appspot.com/static/images/UVic-mark.png" alt="UVic Logo" style="height: 40px; margin-right: 10px;">
          Opto-Nose
        </a>
        <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
          <span class="navbar-toggler-icon"></span>
```

```
        </button>

        <div class="collapse navbar-collapse" id="navbarSupportedContent">

          <ul class="navbar-nav me-auto mb-2 mb-lg-0 ms-lg-4">

            <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{
url_for('rgbarray')}}">Image Generator</a></li>

            <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('capture_image')
}}">Scan the Wound</a></li>

            <li class="nav-item"><a class="nav-link active" href="{{ url_for('image_library') }}">Classify the
Type</a></li>

            <li class="nav-item"><a class="nav-link active" href="#!">App-Trainer</a></li>

          </ul>

        </div>

      </div>

    </nav>



    <!-- Main Content Section-->

    <main class="content">

      <section class="py-5">

        <div class="container px-4 px-lg-5 my-5">

          <div class="flashed-messages">

              <!-- Handle flashed messages -->

              {% with messages = get_flashed_messages(with_categories=True) %}

                {% if messages %}

                  <div>

                    {% for category, message in messages %}

                      <div class="alert alert-{{ category }} alert-dismissible fade show" role="alert">

                        {{ message }}

                        <button type="button" class="btn-close" data-bs-dismiss="alert" aria-
label="Close"></button>

                      </div>
```

```
                {% endfor %}
              </div>
            {% endif %}
          {% endwith %}
        </div>

        <div class="image-capture-section">
          <h4>Align the VoC Array</h4>
          <video id="video" autoplay></video>
          <canvas id="canvas" style="display: none;"></canvas>
          <div class="circle-grid">
            <div class="circle" id="circle1"></div>
            <div class="circle" id="circle2"></div>
            <div class="circle" id="circle3"></div>
            <div class="circle" id="circle4"></div>
            <div class="circle" id="circle5"></div>
            <div class="circle" id="circle6"></div>
            <div class="circle" id="circle7"></div>
            <div class="circle" id="circle8"></div>
            <div class="circle" id="circle9"></div>
          </div>
          <button id="captureButton" class="btn">Capture Image</button>
          <div class="flash-message" id="flashMessage">Alignment Successful! Ready to Capture.</div>
          <div class="button-container" style="margin-top: 20px;">
            <input type="file" id="fileInput" accept="image/*" class="form-control" style="display: inline-
block; width: auto;">
            <button id="uploadButton" class="btn" style="margin-left: 10px;">Upload Image</button>
          </div>
```

```
        </div>
      </div>
    </section>
</main>

<!-- Footer-->
<footer class="bg-dark" style="position: fixed; bottom: 0; width: 100%; z-index: 1000; padding: 30px 0;">
    <div class="container">
        <p class="m-0 text-center text-white">Copyright &copy; Dr. Akbari's Lab</p>
    </div>
</footer>

<!-- JavaScript for Camera Access and Circle Detection -->
<script>
    const video = document.getElementById('video');
    const canvas = document.getElementById('canvas');
    const context = canvas.getContext('2d');
    const circles = document.querySelectorAll('.circle');
    const circlePositions = [
        { x: 60, y: 30 }, { x: 134, y: 30 }, { x: 208, y: 30 },
        { x: 60, y: 104 }, { x: 134, y: 104 }, { x: 208, y: 104 },
        { x: 60, y: 178 }, { x: 134, y: 178 }, { x: 208, y: 178 }
    ];
    const circleRadius = 25;
    const captureButton = document.getElementById('captureButton');
    const flashMessage = document.getElementById('flashMessage');
    const fileInput = document.getElementById('fileInput');
    const uploadButton = document.getElementById('uploadButton');
```

```
let videoStream = null;



// Function to check if a circle region contains color
function isCircleFilled(imageData) {
  const { data, width, height } = imageData;
  let colorPixels = 0;
  const threshold = 0.6;

  for (let i = 0; i < data.length; i += 4) {
    const red = data[i];
    const green = data[i + 1];
    const blue = data[i + 2];
    if (red < 200 || green < 200 || blue < 200) {
      colorPixels++;
    }
  }
  return (colorPixels / (width * height)) > threshold;
}

// Function to capture frame and check alignment
function detectAlignment() {
  canvas.width = video.videoWidth;
  canvas.height = video.videoHeight;
  context.drawImage(video, 0, 0, canvas.width, canvas.height);

  let allAligned = true;
```

```
        circles.forEach((circle, index) => {
            const { x, y } = circlePositions[index];
            const imageData = context.getImageData(x - circleRadius, y - circleRadius, circleRadius * 2,
circleRadius * 2);

            if (isCircleFilled(imageData)) {
                circle.classList.add('aligned');
            } else {
                circle.classList.remove('aligned');
                allAligned = false;
            }
        });

        // Show capture button and flash message if all circles are aligned
        if (allAligned) {
            flashMessage.style.display = 'block';
        } else {
            flashMessage.style.display = 'none';
        }
        return allAligned
    }
    function startVideoStream() {
        navigator.mediaDevices.getUserMedia({ video: true })
            .then(stream => {
            video.srcObject = stream;
            videoStream = stream;  // Store the stream so it can be stopped later
            })
```

```
      .catch(error => {
        console.error('Error accessing the camera:', error);
      });
}
function stopVideoStream() {
  if (videoStream) {
  videoStream.getTracks().forEach(track => track.stop()); // Stop all tracks in the video stream
  }
}
startVideoStream();
function captureAndFreezeFrame() {
  if (video.videoWidth === 0 || video.videoHeight === 0) {
    console.error("Video dimensions are not available.");
    return;
  }

  canvas.width = video.videoWidth;
  canvas.height = video.videoHeight;

  context.drawImage(video, 0, 0, canvas.width, canvas.height);

  video.style.display = 'none';
  canvas.style.display = 'block';

  // Stop the video stream after displaying the frame on the canvas
  setTimeout(() => {
    stopVideoStream();
```

```
      // After 1 second, restart the video stream and hide the canvas
      setTimeout(() => {
        startVideoStream();  // Restart the video stream
        video.style.display = 'block';
        canvas.style.display = 'none';
      }, 1000);
    }, 0);
}
// Capture the image and send it to the backend when the button is clicked
captureButton.addEventListener('click', () => {
    // Draw the current video frame to the canvas
    // Capture the current frame from the video and draw it on the canvas
    if (!detectAlignment()) {
      alert("Please align all circles correctly before capturing.");
      return; // Exit if circles are not aligned, no image is sent to the server
    }

    captureAndFreezeFrame();

    setTimeout(() => {
      canvas.style.display = 'none';
      video.style.display = 'block';
      startVideoStream();
    }, 1000);

    canvas.toBlob((blob) => {
      if (!blob) {
        console.error("Blob creation failed.");
```

```
    alert("Image capture failed. Please try again.");
    return;
}

const formData = new FormData();
formData.append('image', blob, 'captured_image.jpg');  // Append Blob as a file

fetch('/captured_image', {
    method: 'POST',
    body: formData
})
.then(response => {
    if (!response.ok) {
        return response.text().then(text => { throw new Error(`Server error: ${response.status} ${text}`); });
    }
    return response.json();
})
.then(data => {
    if (data.success) {
        console.log("Image sent successfully to the server.");
        alert("Image captured and sent successfully!");
    } else {
        console.log("Image upload failed:", data.error || "No additional error details.");
        alert("Image upload failed.");
    }
})
.catch(error => {
    console.error('Upload error:', error);
```

```
        alert(`Error during upload: ${error.message}`);
      });
  }, 'image/jpeg');
});

// Run detectAlignment at intervals
setInterval(detectAlignment, 200);

uploadButton.addEventListener('click', () => {
  const file = fileInput.files[0];  // Get the selected file from the input

  if (!file) {
    alert('Please select an image file first.');
    return;
  }
  const formData = new FormData();
  formData.append('image', file);  // Append the file to FormData

  fetch('/captured_image', {
    method: 'POST',
    body: formData
  })
  .then(response => {
    if (!response.ok) {
      return response.text().then(text => { throw new Error(`Server error: ${response.status} ${text}`); });
    }
    return response.json();
  })
```

```
        .then(data => {
          if (data.success) {
            console.log("Image sent successfully to the server.");
            alert("Image uploaded successfully!");
          } else {
            console.log("Image upload failed:", data.error || "No additional error details.");
            alert("Image upload failed.");
          }
        })
        .catch(error => {
          console.error('Upload error:', error);
          alert(`Error during upload: ${error.message}`);
        });
      });


  </script>
</body>
</html>
```

Front End: Image Library and Classification Page

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="utf-8" />
  <meta name="viewport" content="width=device-width, initial-scale=1, shrink-to-fit=no" />
  <title>Mostafa's Application</title>
  <!-- Favicon -->
  <link rel="icon" type="image/png" href="{{ url_for('static', filename='images/Martlet-blue.png') }}" />
```

```
    <!-- Bootstrap icons -->
    <link href="https://cdn.jsdelivr.net/npm/bootstrap-icons@1.5.0/font/bootstrap-icons.css" rel="stylesheet"
/>
    <!-- Core theme CSS (includes Bootstrap) -->
    <link rel="stylesheet" href="{{ url_for('static', filename='css/styles.css') }}">
    <style>
      .navbar {
        padding-top: 20px;
        padding-bottom: 20px;
        font-size: 1.2rem;
      }
      .image-library-section {
        display: flex;
        flex-direction: column;
        gap: 20px;
        margin-top: 20px;
      }
      .image-item {
        display: flex;
        align-items: center;
        justify-content: space-between;
        border: 1px solid #ddd;
        padding: 15px;
        border-radius: 8px;
        background-color: #f8f9fa;
      }
      .image-thumbnail {
        width: 50%; /* Adjust the image width to half the container */
```

```
    height: auto; /* Maintain aspect ratio */
    border-radius: 8px;
}
.preprocess-button {
    background-color: #007bff;
    color: #fff;
    border: none;
    padding: 15px 30px;
    cursor: pointer;
    border-radius: 5px;
    font-size: 16px;
    margin-left: 20px;
}
.preprocess-button:hover {
    background-color: #0056b3;
}
.classify-button {
    background-color: #28a745; /* Green color for differentiation */
    color: #fff;
    border: none;
    padding: 15px 30px;
    cursor: pointer;
    border-radius: 5px;
    font-size: 16px;
    margin-left: 20px;
}
.classify-button:hover {
    background-color: #218838; /* Slightly darker green on hover */
```

```
}
.delete-button {
  background-color: #dc3545;
  color: white;
  border: none;
  padding: 10px 20px;
  cursor: pointer;
  border-radius: 5px;
  font-size: 14px;
  margin-left: 10px;
  margin-top: 5px;
}
.delete-button:hover {
  background-color: #c82333;
}
.back-to-top:hover {
  background-color: #0056b3;
}
body {
display: flex;
flex-direction: column;
min-height: 100vh;
}
main {
  flex: 1;
  padding-bottom: 100px;
}
footer {
```

```
    position: relative;
    bottom: 0;
    width: 100%;
    z-index: 1000;
}
.image-item {
    display: flex;
    align-items: center;
    justify-content: space-between;
    padding: 15px;
    background-color: #f8f9fa;
    border: 1px solid transparent; /* Prevent border covering */
    border-radius: 8px;
    margin-bottom: 20px; /* Ensure space between the image and footer */
}

.back-to-top {
    position: fixed;
    bottom: 40px;
    right: 40px;
    display: none;
    background-color: #007bff;
    color: white;
    padding: 10px 20px;
    border-radius: 5px;
    text-align: center;
    cursor: pointer;
    z-index: 1999;
```

```
    }
  </style>
</head>
<body>
  <!-- Navigation -->
  <nav class="navbar navbar-expand-lg navbar-light bg-light">
    <div class="container px-4 px-lg-5">
      <a class="navbar-brand" href="{{ url_for('home') }}">
        <img src="https://storage.googleapis.com/mostafasapp01.appspot.com/static/images/UVic-mark.png" alt="UVic Logo" style="height: 40px; margin-right: 10px;">
        Opto-Nose
      </a>
      <button class="navbar-toggler" type="button" data-bs-toggle="collapse" data-bs-target="#navbarSupportedContent" aria-controls="navbarSupportedContent" aria-expanded="false" aria-label="Toggle navigation">
        <span class="navbar-toggler-icon"></span>
      </button>
      <div class="collapse navbar-collapse" id="navbarSupportedContent">
        <ul class="navbar-nav me-auto mb-2 mb-lg-0 ms-lg-4">
          <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('rgbarray') }}">Image Generator</a></li>
          <li class="nav-item"><a class="nav-link active" aria-current="page" href="{{ url_for('capture_image') }}">Scan the Wound</a></li>
          <li class="nav-item"><a class="nav-link active" href="{{ url_for('image_library') }}">Classify the Type</a></li>
          <li class="nav-item"><a class="nav-link active" href="#!">App-Trainer</a></li>
        </ul>
      </div>
    </div>
  </nav>
```

```
  <!-- Flashed Messages -->
  <div class="container">
    {% with messages = get_flashed_messages(with_categories=True) %}
      {% if messages %}
        <div class="flashed-messages">
          {% for category, message in messages %}
            <div class="alert alert-{{ category }} alert-dismissible fade show mt-3" role="alert">
              {{ message }}
              <button type="button" class="btn-close" data-bs-dismiss="alert" aria-label="Close"></button>
            </div>
          {% endfor %}
        </div>
      {% endif %}
    {% endwith %}
  </div>

  <!-- Main content -->
  <main class="content">
    <section class="container py-5">
      <h2 class="text-center">Uploaded Images</h2>
      <div class="image-library-section">
        <!-- Placeholder for dynamically loaded images -->
        {% for image in uploaded_images %}
        <div class="image-container" id="image-item-{{ image.id }}">
          <img src="{{ image.url }}" alt="Uploaded Image" class="image-thumbnail" id="image-{{ image.id }}">
          <button class="preprocess-button" onclick="preprocessImage('{{ image.id }}', '{{ image.url }}')">Pre-Process</button>
          <button class="delete-button" onclick="deleteImage('{{ image.id }}', '{{ image.url }}')">Delete Image</button>
```

```
        </div>
        {% endfor %}
      </div>
    </section>
</main>

<!-- Back to Top Button -->
<button onclick="scrollToTop()" class="back-to-top btn btn-primary">Back to Top</button>



<!-- Footer -->
<footer class="py-5 bg-dark">
    <div class="container">
        <p class="m-0 text-center text-white">Copyright &copy; Dr. Akbari's Lab</p>
    </div>
</footer>

<!-- Bootstrap core JS -->
<script src="https://cdn.jsdelivr.net/npm/bootstrap@5.2.3/dist/js/bootstrap.bundle.min.js"></script>
<!-- Core theme JS -->
<script src="{{ url_for('static', filename='js/scripts.js') }}"></script>

<!-- Back to Top Button Script -->
<script>
    window.onscroll = function() { toggleBackToTopButton(); };

    function toggleBackToTopButton() {
        const button = document.querySelector('.back-to-top');
```

```
    if (document.body.scrollTop > 50 || document.documentElement.scrollTop > 50) {
        button.style.display = "block";
    } else {
        button.style.display = "none";
    }
}

function scrollToTop() {
    window.scrollTo({ top: 0, behavior: 'smooth' });
}

function preprocessImage(imageId, imageUrl) {
    // Disable the button while processing
    const button = document.querySelector(`#image-item-${imageId} .preprocess-button`);
    button.textContent = "Processing...";
    button.disabled = true;

    // Send the image ID and URL to the server
    fetch('/process_image', {
        method: 'POST',
        body: JSON.stringify({ imageId: imageId, imageUrl: imageUrl }),
        headers: {
            'Content-Type': 'application/json',
        },
    })
        .then(response => response.json())
        .then(data => {
            // Update the image source with the processed image URL
```

```
            const imgElement = document.querySelector(`#image-${imageId}`);
            imgElement.src = data.processedImageUrl;

            const rgbValuesContainer = document.querySelector(`#rgb-values-${imageId}`);

            // Re-enable the button
            button.textContent = "Pre-Process Again";
            button.disabled = false;

            let classifyButton = document.querySelector(`#image-item-${imageId} .classify-button`);
            if (!classifyButton) {
              classifyButton = document.createElement('button');
              classifyButton.textContent = "Classify Image";
              classifyButton.classList.add('classify-button');
              classifyButton.onclick = () => classifyImage(imageId, data.rgbValues);
              button.parentNode.appendChild(classifyButton);
            }
          })
          .catch(error => {
            console.error("Error processing image:", error);
            button.textContent = "Pre-Process";
            button.disabled = false;
          });
      }
      function deleteImage(imageId, imageUrl) {
        // Send the delete request to the backend
        fetch('/delete_image', {
          method: 'POST',
```

```
        body: JSON.stringify({ imageId: imageId, imageUrl: imageUrl }),
        headers: {
          'Content-Type': 'application/json',
        },
      })
        .then(response => response.json())
        .then(data => {
          if (data.success) {
            // Remove the image container from the DOM
            const imageItem = document.querySelector(`#image-item-${imageId}`);
            imageItem.remove();
            console.log("Image deleted successfully.");
          } else {
            console.error("Failed to delete image:", data.error);
          }
        })
        .catch(error => {
          console.error("Error deleting image:", error);
        });
    }
    function classifyImage(imageId, rgbValues) {
      // Disable the button while processing
      const button = document.querySelector(`#image-item-${imageId} .classify-button`);
      button.textContent = "Classifying...";
      button.disabled = true;

      // Send the image ID and URL to the server
      fetch('/classify_image', {
```

```
            method: 'POST',
            body: JSON.stringify({ imageId: imageId, rgbValues: rgbValues }),
            headers: {
              'Content-Type': 'application/json',
            },
          })
            .then(response => response.json())
            .then(data => {
              // Display classification result
              alert(`Classification Result: ${data.result}`);
              button.textContent = "Classify Image";
              button.disabled = false;
            })
            .catch(error => {
              console.error("Error classifying image:", error);
              button.textContent = "Classify Image";
              button.disabled = false;
            });
        }
    </script>
</body>
</html>
```

Backend: Routes

```
from flask import render_template, request, redirect, url_for, flash, jsonify
from werkzeug.utils import secure_filename
import requests
from opto import app
```

```
from PIL import Image, ImageDraw, ImageOps, ImageEnhance
import numpy as np
from google.cloud import storage
from google.oauth2 import service_account
import io
import tempfile
import time
import logging
import cv2
from math import sqrt
from matplotlib import pyplot as plt
from sklearn.cluster import KMeans
from tensorflow.keras.models import load_model


# Config for file upload
BUCKET_NAME = 'mostafasapp01.appspot.com'
MODEL_PATH_IN_GCS = 'models/optimized_ann_model.h5'
LOCAL_MODEL_PATH = '/tmp/optimized_ann_model.h5'


model = None


ENCODED_TO_DECODED_CLASS_MAPPING = {
  0: "EC",
  1: "EC+PA",
  2: "EC+PA+SA",
  3: "EC+SA",
  4: "PA",
  5: "SA",
```

```
    6: "SA + PA",
    7: "SA+PA"
}

def download_credentials_from_gcs(bucket_name, source_blob_name, destination_file_name):
    """Downloads the credentials file from Google Cloud Storage without credentials."""
    # This uses unauthenticated access to download the file first
    storage_client = storage.Client()  # No credentials needed here
    bucket = storage_client.bucket(bucket_name)
    blob = bucket.blob(source_blob_name)
    blob.download_to_filename(destination_file_name)
    print(f"Downloaded credentials to {destination_file_name}")

def download_model_from_gcs():
    """Downloads the model from GCS to the local file system."""
    storage_client = storage.Client()
    bucket = storage_client.bucket(BUCKET_NAME)
    blob = bucket.blob(MODEL_PATH_IN_GCS)
    blob.download_to_filename(LOCAL_MODEL_PATH)

def load_model_once(force_reload=False):
    """Loads the model into memory."""
    global model
    if model is None or force_reload:
        download_model_from_gcs()  # Download the model only once
        model = load_model(LOCAL_MODEL_PATH)
        print("Model loaded successfully!")
```

```
def upload_image_to_gcs(image, destination_blob_name):
    """Uploads a file to Google Cloud Storage."""
    bucket = storage_client.bucket(BUCKET_NAME)
    blob = bucket.blob(destination_blob_name)

    # Quality enhancement
    enhanced_image = enhance_image(image)

    # Convert PIL Image to bytes for uploading
    image_bytes = io.BytesIO()
    enhanced_image.save(image_bytes, format='JPEG', quality = 95, dpi=(300, 300))
    image_bytes.seek(0)

    blob.upload_from_file(image_bytes, content_type='image/jpeg')

    blob.reload()

    # Upload to GCS
    blob.make_public()

    # Return the public URL of the uploaded file
    return f"https://storage.googleapis.com/{BUCKET_NAME}/{destination_blob_name}"

def upload_image(image, destination_blob_name):
    """Uploads a file to Google Cloud Storage."""
    bucket = storage_client.bucket(BUCKET_NAME)
    blob = bucket.blob(destination_blob_name)
```

```
    blob.upload_from_file(image, content_type='image/jpeg')

    blob.reload()

    # Upload to GCS
    blob.make_public()

    # Return the public URL of the uploaded file
    return f"https://storage.googleapis.com/{BUCKET_NAME}/{destination_blob_name}"

def get_uploaded_images():
    """Fetches the urls from the GCS bucket."""
    storage_client = storage.Client()
    bucket = storage_client.bucket(BUCKET_NAME)
    blobs = bucket.list_blobs(prefix='static/voc/')

    image_urls = []
    for blob in blobs:
        if not blob.name.endswith('/'):  # Exclude folder paths
            blob.make_public()
            image_urls.append({'url': blob.public_url})

    return image_urls if image_urls else []

# Temporary credentials file to store downloaded service account credentials
tmp_credentials_file = tempfile.NamedTemporaryFile(delete=False)

# Step 1: Download the credentials from GCS using unauthenticated access
```

```
download_credentials_from_gcs(BUCKET_NAME, 'credentials/mostafasapp01-a4fa5590532f.json',
tmp_credentials_file.name)


load_model_once()


# Step 2: Load the downloaded credentials for authenticated access

credentials = service_account.Credentials.from_service_account_file(tmp_credentials_file.name)


# Now we can initialize the storage client with credentials

storage_client = storage.Client(credentials=credentials)


def enhance_image(image):

  # Step 1: Adjust brightness (increase to 1.0 times the original)

  brightness_enhancer = ImageEnhance.Brightness(image)

  image_bright = brightness_enhancer.enhance(0.8)  # slight brightness increase


  # Step 2: Adjust saturation (increase to 1.5 times the original)

  saturation_enhancer = ImageEnhance.Color(image_bright)

  image_saturated = saturation_enhancer.enhance(1.2)  # slight saturation


  contrast_enhancer = ImageEnhance.Contrast(image_saturated)

  image_contrasted = contrast_enhancer.enhance(1.3)  #slight contrast

  return image_contrasted


def adjust_brightness_and_contrast(color, brightness_factor=1.5, contrast_factor=1.2, threshold=50):

  avg_brightness = sum(color) / 3  # Calculate average brightness of the RGB color

  if avg_brightness < threshold:

    # Apply brightness adjustment
```

```
        brightened_color = [min(int(c * brightness_factor), 255) for c in color]

        # Apply contrast adjustment (boost contrast by shifting mid-range colors)
        adjusted_color = [int((c - 128) * contrast_factor + 128) for c in brightened_color]
        adjusted_color = [min(max(c, 0), 255) for c in adjusted_color]

        return tuple(adjusted_color)
    return color  # Return original color if it's already bright

def generate_image_from_array(rgb_array):
    """Generates an image after user enters the data in the table"""
    rect_width = 100
    rect_height = 100
    circle_radius = 30
    background_color = (0, 0, 0)
    border_color = (0, 0, 0)
    border_size = 20

    rows = len(rgb_array) // 3
    img_width = 3 * rect_width
    img_height = rows * rect_height
    image = Image.new("RGB", (img_width, img_height), background_color)
    draw = ImageDraw.Draw(image)

    for i, color in enumerate(rgb_array):

        adjusted_color = adjust_brightness_and_contrast(color)
        top_left_x = (i % 3) * rect_width
```

```
        top_left_y = (i // 3) * rect_height
        circle_bbox = [
            (top_left_x + rect_width // 2 - circle_radius, top_left_y + rect_height // 2 - circle_radius),
            (top_left_x + rect_width // 2 + circle_radius, top_left_y + rect_height // 2 + circle_radius)
        ]
        draw.ellipse(circle_bbox, fill=tuple(adjusted_color))

    image_with_border = ImageOps.expand(image, border=border_size, fill=border_color)
    unique_filename = f"output_image_{int(time.time())}.jpg"
    gcs_image_path = f"static/images/{unique_filename}"

    # Upload the image to GCS and return the URL
    image_url = upload_image_to_gcs(image_with_border, gcs_image_path)

    return image_url

def extract_accurate_center_rgb(region):
    if region is None or region.size == 0:
        print("Error: The region is empty.")
        return None

    # Get the dimensions of the region
    height, width, _ = region.shape

    # Find the center coordinates
    center_y = height // 2
    center_x = width // 2
```

```
    # Extract the BGR value at the center
    center_bgr = region[center_y, center_x]

    return tuple(center_bgr)

def extract_all_rgb_values(image):
    """Extracts all RGB values from the image and returns them as an array."""
    # Flatten the image array and get RGB values
    pixels = image.reshape(-1, 3)  # Reshape to N x 3, where N is the number of pixels and 3 represents RGB
    return pixels


def find_optimal_clusters(data, max_clusters=10):
    """Finds the optimal number of clusters using the Elbow Method or Silhouette Analysis."""
    inertia_values = []
    silhouette_scores = []
    for num_clusters in range(2, max_clusters + 1):
        kmeans = KMeans(n_clusters=num_clusters)
        kmeans.fit(data)
        inertia_values.append(kmeans.inertia_)

        # Calculate silhouette score for each cluster
        score = silhouette_score(data, kmeans.labels_)
        silhouette_scores.append(score)

    # Determine the optimal number of clusters based on the highest silhouette score
    optimal_clusters = np.argmax(silhouette_scores) + 2  # Adding 2 because range starts at 2
```

```
    return optimal_clusters


def cluster_colors(rgb_values, image_shape, max_clusters=10):
    """Clusters the RGB values into specified number of clusters to define dominant colors."""
    #optimal_clusters = find_optimal_clusters(rgb_values, max_clusters=max_clusters)
    #print("the optimal clusters are:", optimal_clusters)
    kmeans = KMeans(n_clusters=5)
    kmeans.fit(rgb_values)
    labels = kmeans.labels_.reshape(image_shape[:2])
    return kmeans.cluster_centers_, labels



def display_image_with_scaled_coordinates(data, canvas_height, canvas_width, scale_factor=1,
canvas_color=(0, 0, 0), circle_radius=10):
    # Scale the coordinates while maintaining relative spacing
    scaled_data = [(int(x * scale_factor), int(y * scale_factor), r, g, b) for x, y, r, g, b in data]

    # Create a black canvas with the given dimensions
    image = Image.new("RGB", (canvas_width, canvas_height), canvas_color)
    draw = ImageDraw.Draw(image)

    # Draw circles at the scaled coordinates with the given RGB values
    for x, y, r, g, b in scaled_data:
        circle_bbox = [
            (x - circle_radius, y - circle_radius),
            (x + circle_radius, y + circle_radius)
        ]
        draw.ellipse(circle_bbox, fill=(r, g, b))
```

```
    # Display the image
    plt.figure(figsize=(canvas_width/25 , canvas_height/25 ))  # Adjust display size for large/small images
    plt.imshow(image)
    plt.axis("off")
    plt.show()


def preprocess_and_extract_rgb_from_image(image):
    # Convert the image to the correct format (if it's passed as a raw file)
    nparr = np.frombuffer(image, np.uint8)
    img = cv2.imdecode(nparr, cv2.IMREAD_COLOR)

    # Ensure image is loaded properly
    if img is None:
        print("Error: Could not load the image from the buffer.")
        return []

    # Save the original image for later RGB extraction
    original_img = img.copy()

    # Extract all RGB values directly from the image
    rgb_values_all = extract_all_rgb_values(img)

    # Cluster the colors to get the dominant colors and labels for each pixel
    dominant_colors, labels = cluster_colors(rgb_values_all, img.shape)

    output_img = img.copy()
```

```
# Create a blank mask for drawing detected regions
mask = np.zeros(img.shape[:2], dtype="uint8")

# Loop over each cluster and identify areas of dominant color
all_rgb_values = []
processed_regions = []
all_rgb_coordinates = []
for cluster_id in range(dominant_colors.shape[0]):
  # Mask the areas corresponding to the current cluster
  cluster_mask = np.where(labels == cluster_id, 255, 0).astype("uint8")

  # Find contours of the clustered regions
  contours, _ = cv2.findContours(cluster_mask, cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

  # Filter out non-circular contours and adjust the area range to detect circles
  min_area = 100  # Adjust the minimum area as needed
  max_area = 12000  # Adjust the maximum area as needed

  for i, contour in enumerate(contours):
    area = cv2.contourArea(contour)
    perimeter = cv2.arcLength(contour, True)
    if perimeter == 0:
      continue
    circularity = 4 * np.pi * (area / (perimeter ** 2))
    if 0.1 < circularity <= 1.5 and min_area < area < max_area:
      x, y, w, h = cv2.boundingRect(contour)
      is_duplicate = False
      for (px, py, pw, ph) in processed_regions:
```

```
            if (x < px + pw and x + w > px and y < py + ph and y + h > py):
                is_duplicate = True
                break

        if is_duplicate:
            continue  # Skip this region if it's a duplicate

        # Mark this region as processed
        processed_regions.append((x, y, w, h))
        center_x = x + w // 2
        center_y = y + h // 2
        cv2.rectangle(output_img, (x, y), (x + w, y + h), (0, 255, 0), 2)

        # Draw on the mask for visualization
        cv2.drawContours(mask, [contour], -1, 255, -1)

        # Extract a zoomed-in region
        zoomed_region = original_img[y:y+h, x:x+w]

        # Extract accurate mean RGB from the center of the zoomed-in region
        mean_bgr = extract_accurate_center_rgb(zoomed_region)
        mean_rgb = (mean_bgr[2], mean_bgr[1], mean_bgr[0])  # Convert BGR to RGB

        all_rgb_coordinates.append((center_x, center_y, *mean_rgb))
        all_rgb_values.append((x, y, mean_rgb))

# Use a threshold to determine if two regions are in the same row
row_threshold = 10
```

```
# Group regions by row and then sort within each row
all_rgb_values.sort(key=lambda region: region[1])
sorted_regions = []
current_row = []
for i, region in enumerate(all_rgb_values):
    if i == 0:
        current_row.append(region)
    else:
        # Check if the current region is in the same row as the previous one
        if abs(region[1] - current_row[-1][1]) < row_threshold:
            current_row.append(region)
        else:
            # Sort the current row by x coordinate and add to sorted_regions
            current_row.sort(key=lambda r: r[0])
            sorted_regions.extend(current_row)
            current_row = [region]
# Add the last row to sorted regions
if current_row:
    current_row.sort(key=lambda r: r[0])
    sorted_regions.extend(current_row)

# Now sorted_regions contains the regions sorted from top-left to bottom-right,

for idx, (x, y, rgb) in enumerate(sorted_regions, start=1):
    cv2.putText(output_img, str(idx), (x + w // 2, y + h // 2), cv2.FONT_HERSHEY_SIMPLEX, 0.5, (0, 0, 255), 1)

output_image_height, output_image_width = output_img.shape[:2]
```

```
    rgb_values = [{"id": idx, "rgb": rgb} for idx, (x, y, rgb) in enumerate(sorted_regions, start=1)]

    return rgb_values, output_img

def convert_rgb_dict_to_array(rgb_dict):
    max_regions = 9  # Total expected regions
    flat_rgb_array = []
    rgb_dict = {item['id']: item['rgb'] for item in rgb_dict}

    # Ensure the dictionary keys are processed in ascending order
    for i in range(1, max_regions + 1):
        if i in rgb_dict:
            # Add RGB values to the array
            flat_rgb_array.extend(rgb_dict[i])  # Add the tuple values directly
        else:
            # If region is missing, add padding (three zeros)
            flat_rgb_array.extend([0, 0, 0])

    return flat_rgb_array


@app.route("/", methods=["GET", "POST"])
def home():
    return render_template("home.html", image_path=None)

@app.route("/rgbimg", methods=["GET", "POST"])
def rgbarray():
```

```
    return render_template("rgbarray.html", image_path=None)


@app.route('/process_table', methods=['POST'])
def process_table():
    try:
        dyes = []
        rgb_values = []

        # Validate the table data
        for i in range(9):
            dye_name = request.form.get(f'dye_name_{i}')
            r_value = request.form.get(f'r_{i}')
            g_value = request.form.get(f'g_{i}')
            b_value = request.form.get(f'b_{i}')

            # Check if any value is missing (explicitly check for None or empty strings)
            if dye_name is None or dye_name.strip() == "" or r_value is None or r_value.strip() == "" or g_value is None or g_value.strip() == "" or b_value is None or b_value.strip() == "":
                flash(f"Missing value in row {i + 1}. Please ensure all dye names and RGB values are filled.", "danger")
                return render_template('rgbarray.html', image_path=None)

            try:
                # Convert the RGB values to integers
                r_value = int(r_value)%256
                g_value = int(g_value)%256
                b_value = int(b_value)%256
            except ValueError:
                # If conversion fails, flash an error message
```

```
            flash(f"Invalid RGB value in row {i + 1}. Please enter valid integers.", "danger")
            return render_template('rgbarray.html', image_path=None)

        # Append valid data to lists
        dyes.append(dye_name)
        rgb_values.append([r_value, g_value, b_value])

    # If all data is valid, process the table
    rgb_array = np.array(rgb_values)

    # Generate image from the array and upload to GCS
    image_url = generate_image_from_array(rgb_array)
    flash("Table data processed successfully and image generated!", "success")

    return render_template('rgbarray.html', image_path=image_url, dyes=dyes, rgb_values=rgb_values,
rgb_array=rgb_array)

  except Exception as e:
    logging.error(f"Error occurred: {e}")  # Log the error to the server log for debugging
    flash("An unexpected error occurred. Please try again.", "danger")
    return render_template('rgbarray.html', image_path=None)

@app.route('/captured_image', methods=['POST','GET'])
def capture_image():
  if request.method == 'POST':
    if 'image' not in request.files:
      return jsonify({"success": False, "error": "No image file provided"}), 400
    image_file = request.files['image']
```

```
        if image_file:
            # Optionally, save or process the file here
            unique_filename = f"voc_array_{int(time.time())}.jpg"
            gcs_image_path = f"static/voc/{unique_filename}"
            image_path = upload_image(image_file, gcs_image_path)
        # Respond with success
            return jsonify({"success": True}), 200
        else:
            return jsonify({"success": False, "error": "File not received"}), 400

    return render_template('Capture.html')


@app.route('/image_library', methods=['POST','GET'])
def image_library():
    images = get_uploaded_images() or []
    # Assign a unique ID to each image by adding an 'id' field
    for idx, image in enumerate(images, start=1):
        image['id'] = idx

    # Pass the images with IDs to the template
    return render_template('imglib.html', uploaded_images=images)


@app.route('/process_image', methods=['POST'])
def process_image():
    # Get the image ID from the AJAX request
    data = request.get_json()
```

```
image_id = data.get('imageId')

image_url = data.get('imageUrl')

rgb_values = []


response = requests.get(image_url)

if response.status_code != 200:

   return jsonify({"error": "Failed to download image"}), 400


# Call the preprocess function to process the image

rgb_values, processed_image = preprocess_and_extract_rgb_from_image(response.content)


output_img_rgb = cv2.cvtColor(processed_image, cv2.COLOR_BGR2RGB)


# Save the processed image to a BytesIO buffer using matplotlib

buffer = io.BytesIO()

plt.imshow(output_img_rgb)

plt.axis("off")  # Hide axes for a clean image

plt.savefig(buffer, format='png', bbox_inches='tight', pad_inches=0)

buffer.seek(0)  # Move to the beginning of the file object


timestamp = int(time.time())

destination_blob_name = f"static/preprocessed/{image_id}_{timestamp}_processed.jpg"


processed_image_url = upload_image(buffer, destination_blob_name)


rgb_values = convert_rgb_dict_to_array(rgb_values)


reference_array = [184, 155, 115, 213, 200, 187, 199, 194, 191, 216, 193, 113, 98, 98, 199, 216, 194, 143,
```

```
123, 132, 95, 142, 133, 78, 167, 104, 142]


    rgb_values = [int(rgb - ref) for rgb, ref in zip(rgb_values, reference_array)]


    # Return the updated image URL
    return jsonify({
        "processedImageUrl": processed_image_url,
        "rgbValues": rgb_values
    })


@app.route('/delete_image', methods=['POST'])
def delete_image():
    data = request.get_json()
    image_url = data.get('imageUrl')


    try:
        blob_name = image_url.split(f"{BUCKET_NAME}/")[1]
        bucket = storage_client.bucket(BUCKET_NAME)
        blob = bucket.blob(blob_name)
        blob.delete()


        return jsonify({"success": True})
    except Exception as e:
        logging.error(f"Error deleting image: {e}")
        return jsonify({"success": False, "error": str(e)}), 500


@app.route('/classify_image', methods=['POST'])
def classify_image():
```

```
    data = request.get_json()

    rgb_values = data.get('rgbValues')



    try:

        # Check if the RGB values are in the correct format

        if not rgb_values or len(rgb_values) != 27:  # Expecting 9 regions * 3 (RGB)

            raise ValueError("Invalid RGB values format. Ensure it has exactly 27 values.")



        # Convert RGB values to a NumPy array for the model

        rgb_array = np.array(rgb_values).reshape(1, -1)



        # Predict using the preloaded model

        prediction = model.predict(rgb_array)

        predicted_classes = np.argmax(prediction, axis=1)



        predicted_class = int(predicted_classes) if isinstance(predicted_classes, np.ndarray) else
predicted_classes



        predicted_label = ENCODED_TO_DECODED_CLASS_MAPPING[predicted_class] if predicted_class in
ENCODED_TO_DECODED_CLASS_MAPPING else "Unknown"



        # Return the predicted class

        return jsonify({"result": f"Predicted Classes: {predicted_label}"})

    except Exception as e:

        logging.error(f"Error during classification: {e}")

        return jsonify({"error": str(e)}), 500
```

BackEnd: Run

```
from opto import app

import os
```

```
if __name__ == '__main__':
    app.run(host='0.0.0.0', port = 8080)
```

Requirements modules for the MAK-1 App

```
bcrypt==4.2.0
blinker==1.8.2
cachetools==5.5.0
certifi==2024.8.30
charset-normalizer==3.3.2
click==8.1.7
colorama==0.4.6
Flask==3.0.3
Flask-Bcrypt==1.0.1
Flask-Login==0.6.3
Flask-SQLAlchemy==3.1.1
google-api-core==2.20.0
google-auth==2.35.0
google-auth-httplib2==0.2.0
google-cloud-core==2.4.1
google-cloud-storage==2.18.2
google-crc32c==1.6.0
google-resumable-media==2.7.2
googleapis-common-protos==1.65.0
greenlet==3.1.1
httplib2==0.22.0
idna==3.10
itsdangerous==2.2.0
Jinja2==3.1.4
MarkupSafe==2.1.5
```

matplotlib==3.7.1

oauthlib==3.2.2

opencv-python==4.5.5.62

pillow==10.4.0

proto-plus==1.24.0

pyasn1==0.6.1

pyasn1_modules==0.4.1

pyparsing==3.1.4

requests==2.32.3

requests-oauthlib==2.0.0

rsa==4.9

scikit-learn==1.3.1

SQLAlchemy==2.0.35

tensorflow==2.18.0

typing_extensions==4.12.2

urllib3==2.2.3

Werkzeug==3.0.4